

DR

DeepRob

Lecture 6
Backpropagation
University of Michigan and University of Minnesota

$$\frac{\partial L}{\partial W_{\ell_1}}$$

$$\frac{\partial L}{\partial W_{\ell_2}}$$

$$\frac{\partial L}{\partial W_{\ell_3}}$$

$$\frac{\partial L}{\partial W_{\ell_4}}$$

$$\frac{\partial L}{\partial W_{\ell_5}}$$

$$\frac{\partial L}{\partial \text{Out}}$$



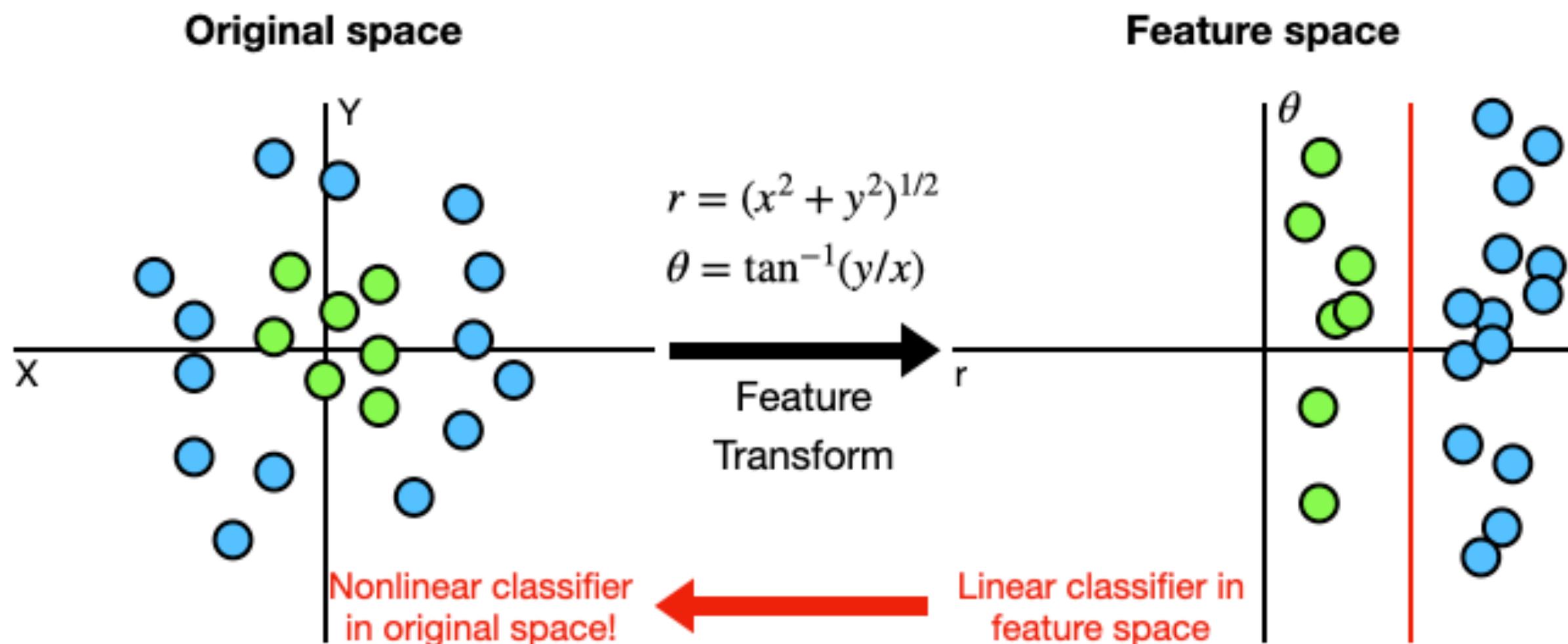
Project 1 – Reminder

- Instructions and code available on the website
 - Here: deeprob.org/projects/project1/
- Uses Python, PyTorch and Google Colab
- Implement KNN, linear SVM, and linear softmax classifiers
- **Autograder is online and updated**
- **Due Thursday, January 26th 11:59 PM EST**

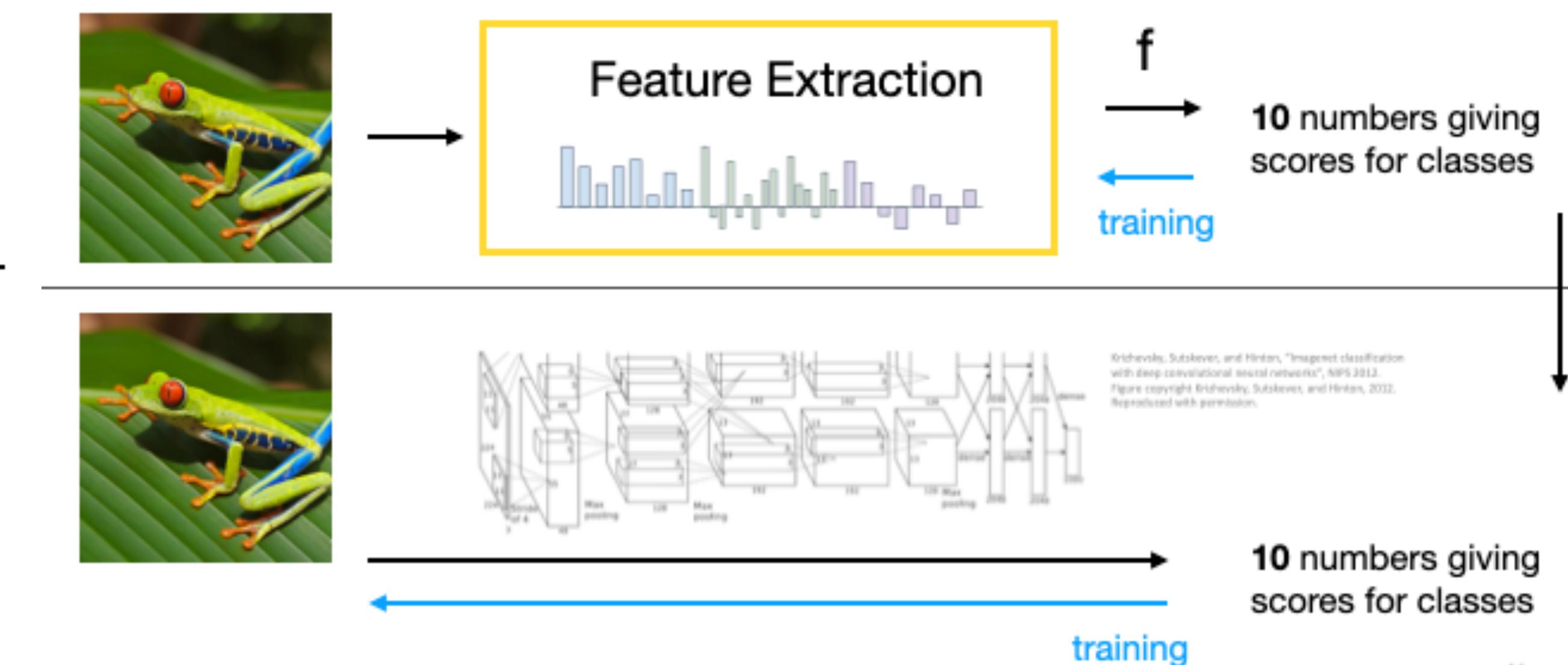


Recap from Previous Lecture

Feature transform + Linear classifier allows nonlinear decision boundaries



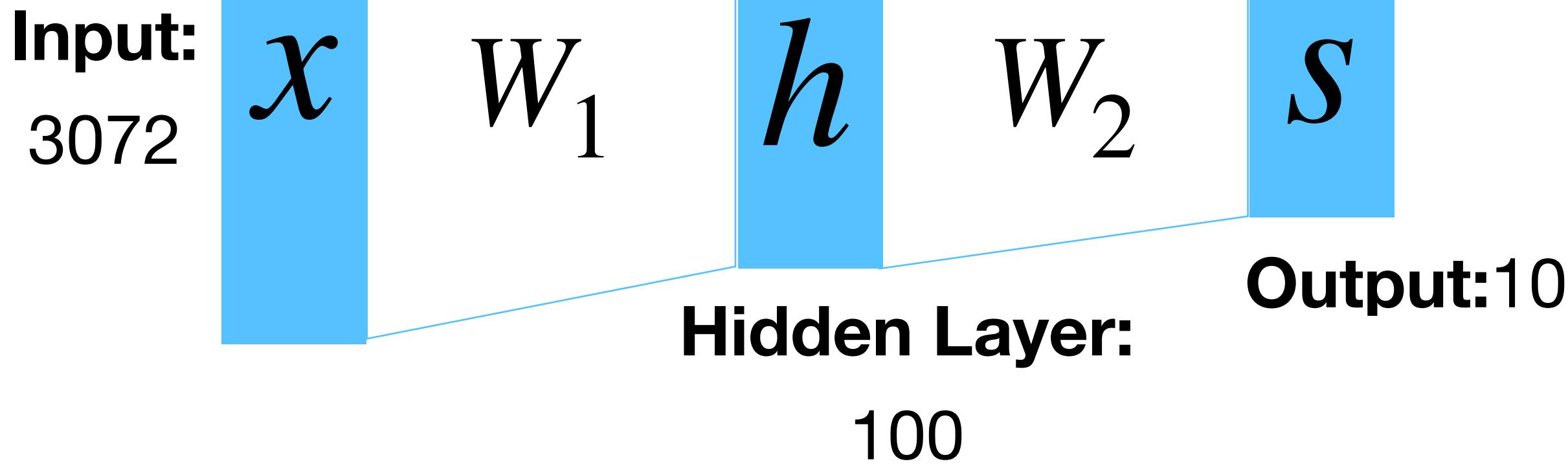
Neural Networks as learnable feature transforms



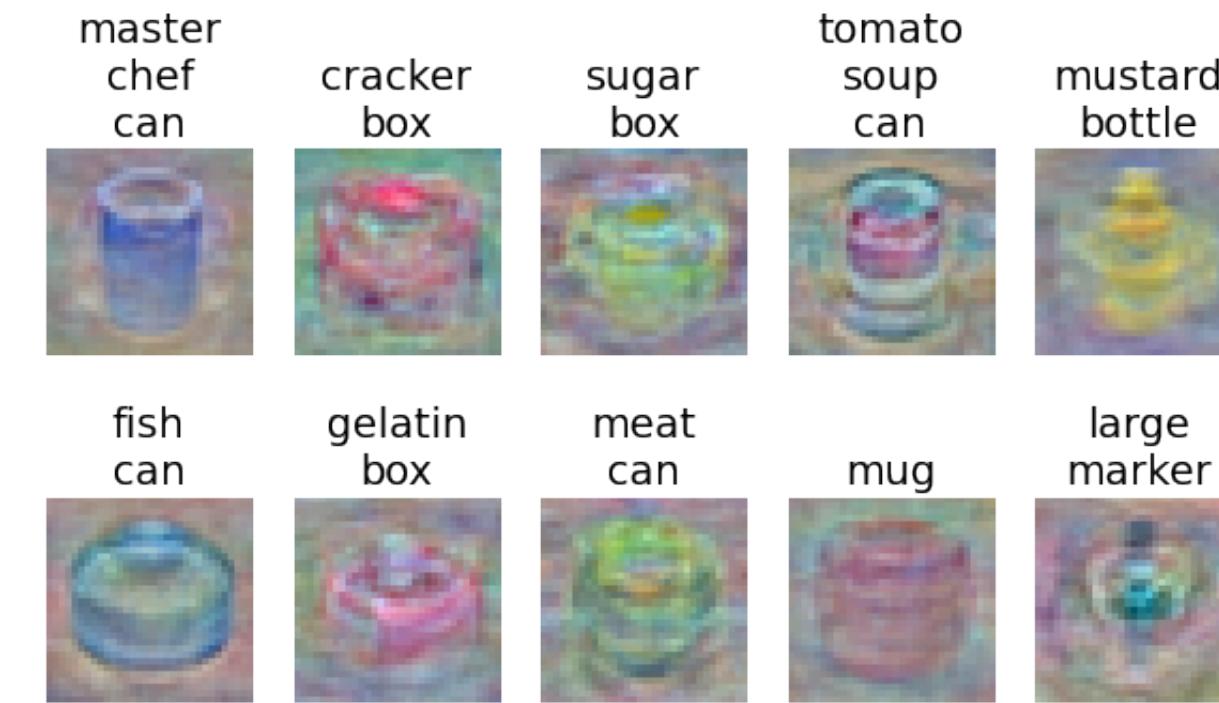
Recap from Previous Lecture

From linear classifiers to
fully-connected networks

$$f(x) = W_2 \max(0, W_1 x + b_1) + b_2$$



Linear classifier: One template per class



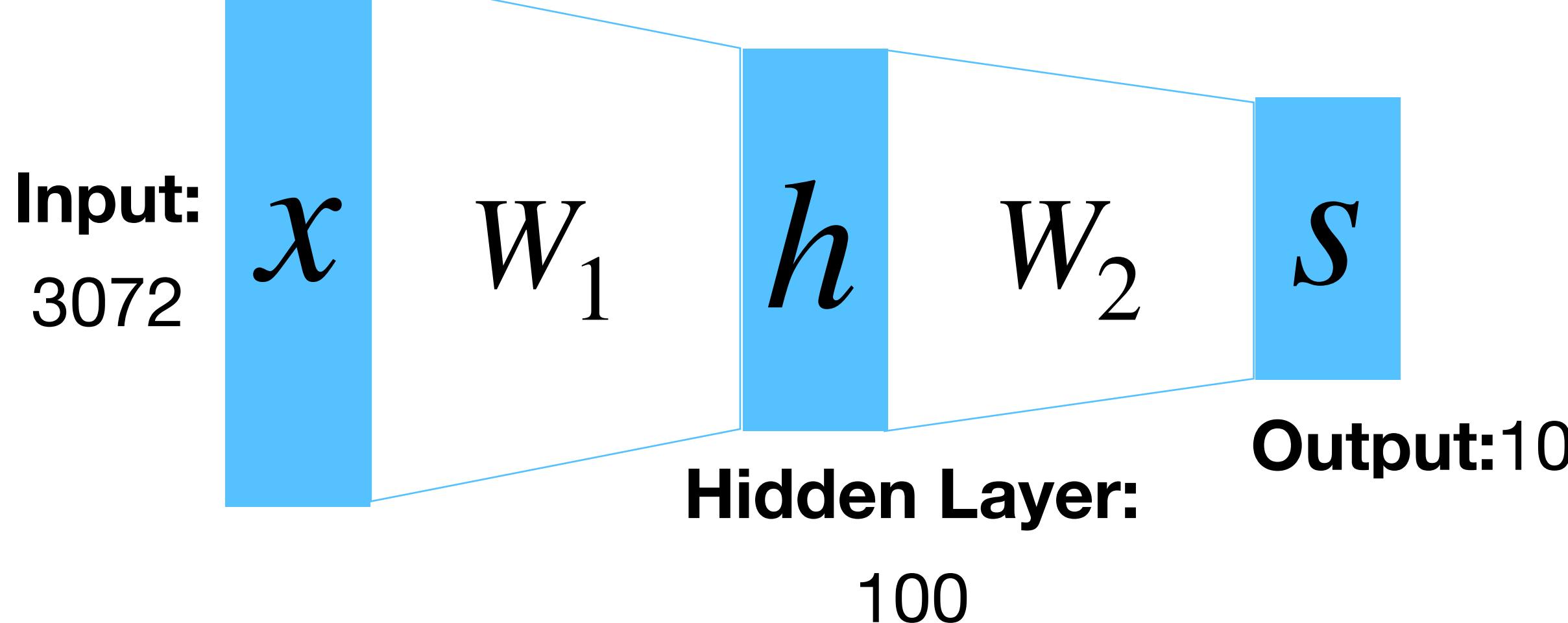
Neural networks: Many reusable templates



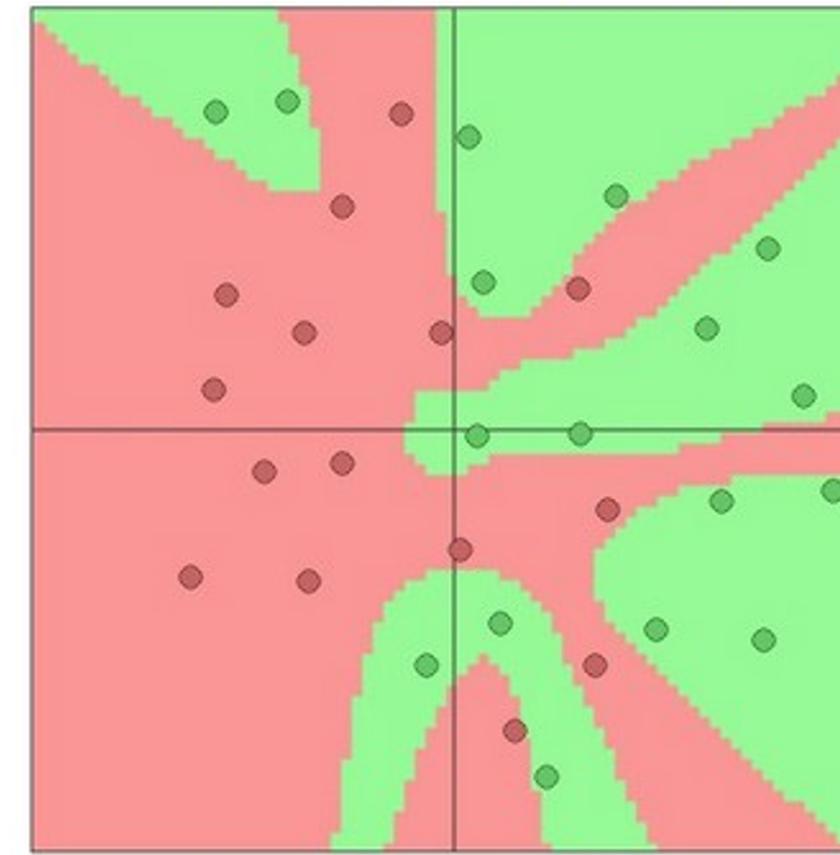
Recap from Previous Lecture

From linear classifiers to
fully-connected networks

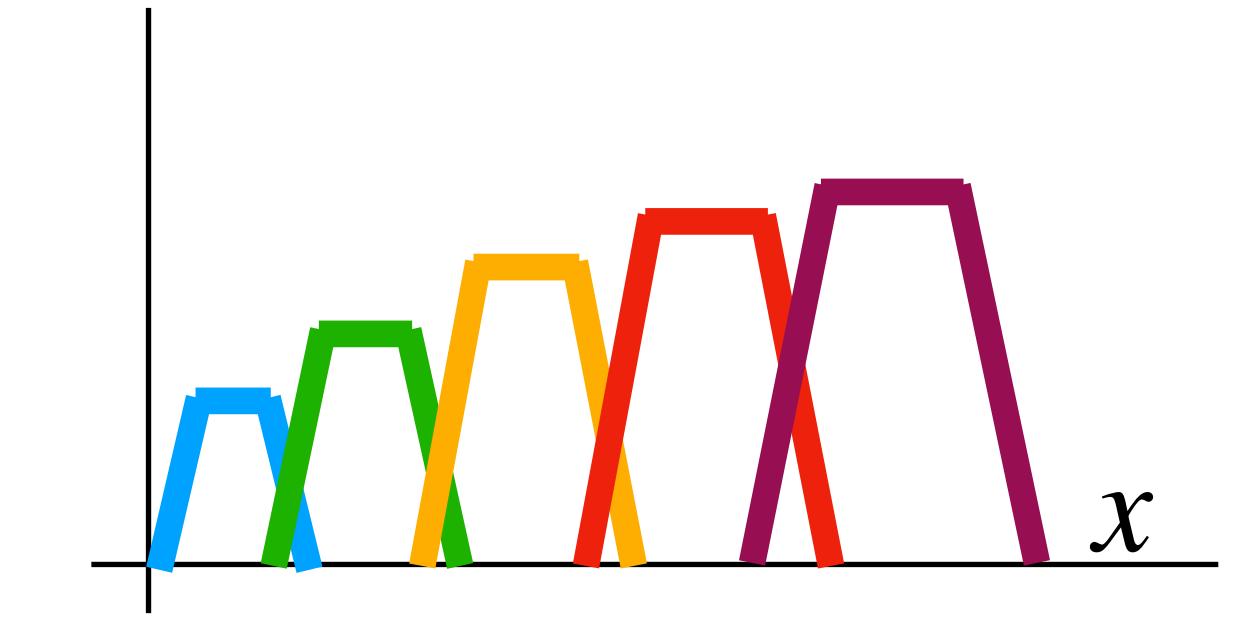
$$f(x) = W_2 \max(0, W_1 x + b_1) + b_2$$



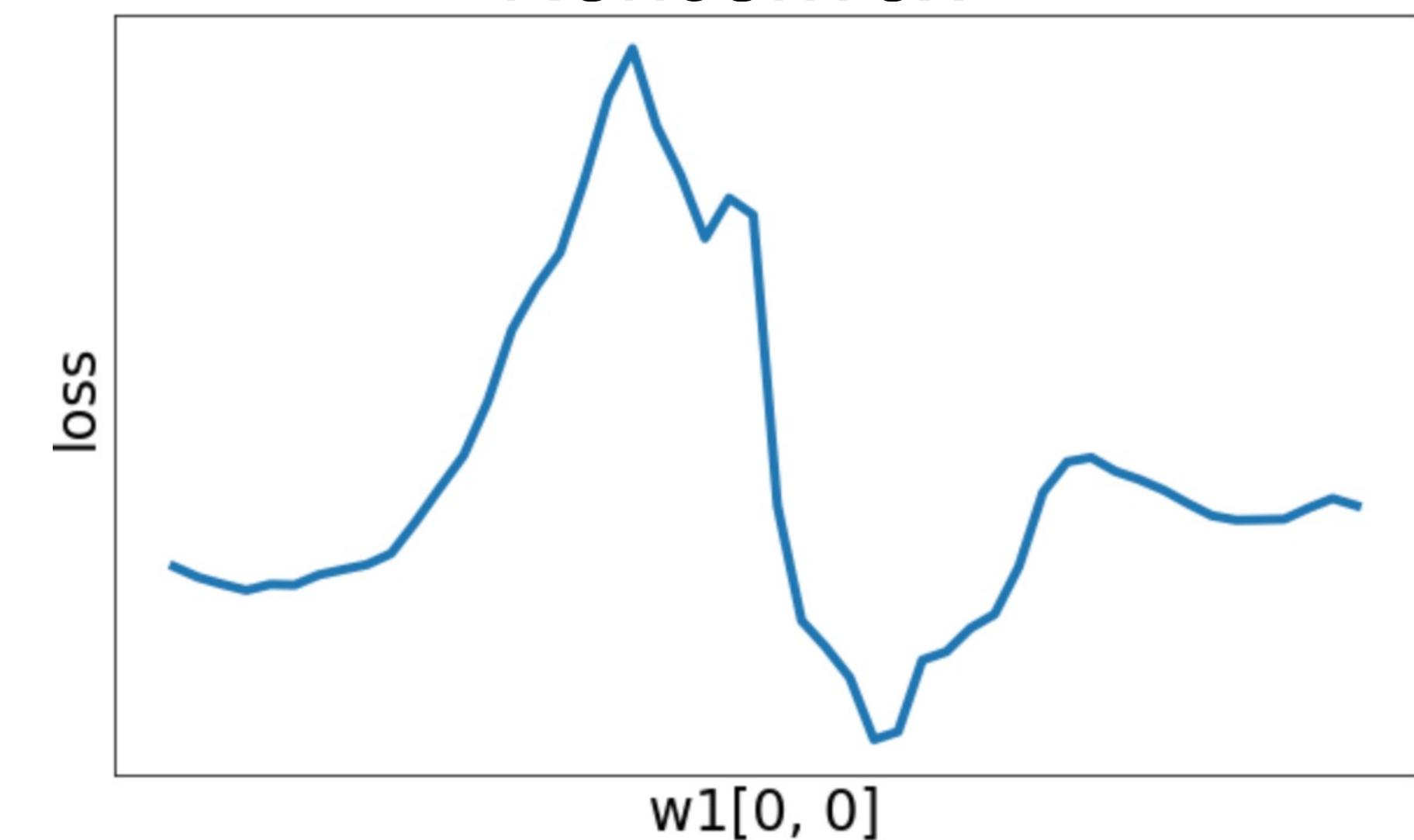
Space Warping



Universal approximation



Nonconvex



Problem: How to compute gradients?

$$s = W_2 \max(0, W_1 x + b_1) + b_2$$

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

$$R(W) = \sum_k W_k^2$$

$$L(W_1, W_2, b_1, b_2) = \frac{1}{N} \sum_{i=1}^N L_i + \lambda R(W_1) + \lambda R(W_2)$$

If we can compute $\frac{\delta L}{\delta W_1}, \frac{\delta L}{\delta W_2}, \frac{\delta L}{\delta b_1}, \frac{\delta L}{\delta b_2}$ then we can optimize with SGD

(Bad) Idea: Derive $\nabla_W L$ on paper

$$s = f(x; W) = Wx$$

$$\begin{aligned} L_i &= \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1) \\ &= \sum_{j \neq y_i} \max(0, W_{j,:} x - W_{y_i,:} x + 1) \\ L &= \frac{1}{N} \sum_{i=1}^N L_i + \lambda \sum_k W_k^2 \\ &= \frac{1}{N} \sum_{i=1}^N \sum_{j \neq y_i} \max(0, W_{j,:} x - W_{y_i,:} x + 1) + \lambda \sum_k W_k^2 \end{aligned}$$

Problem: Very tedious with lots of matrix calculus

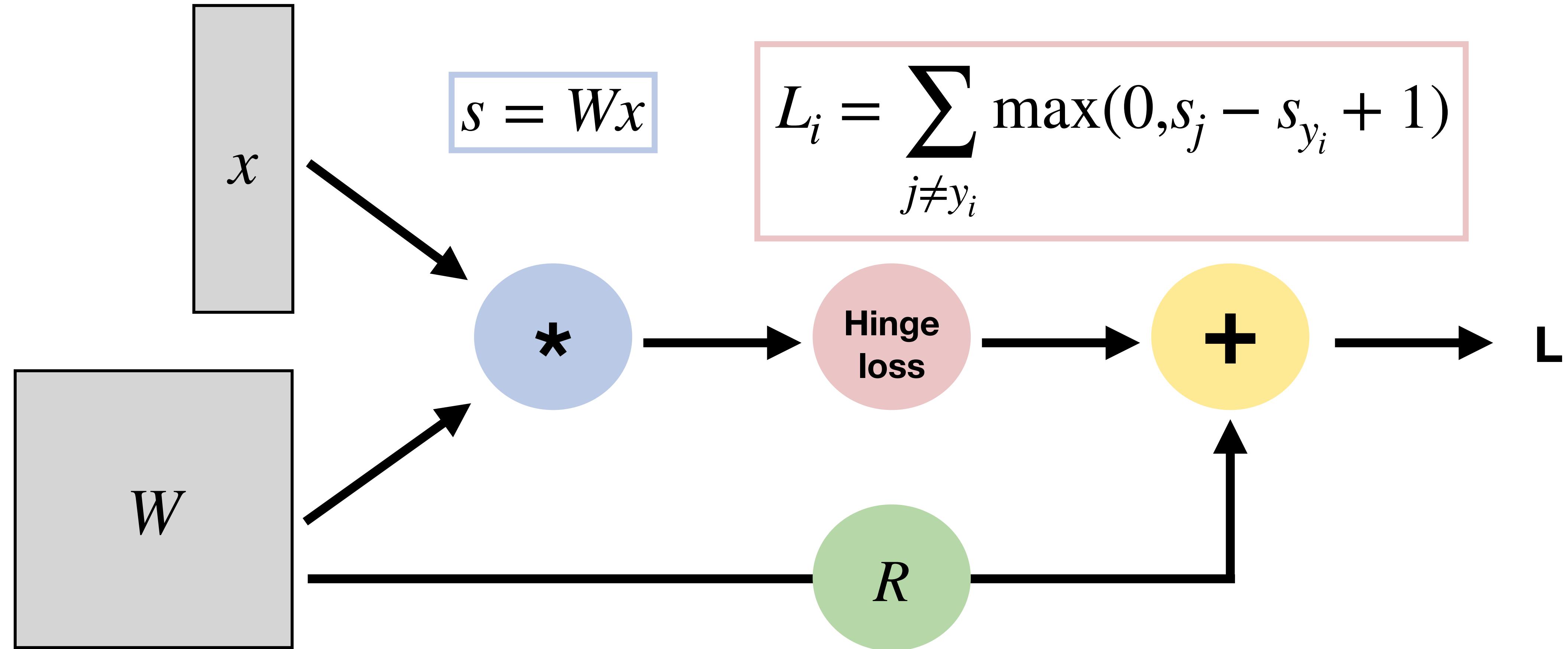
Problem: What if we want to change the loss? E.g. use softmax instead of SVM? Need to re-derive from scratch. Not modular!

Problem: Not feasible for very complex models!

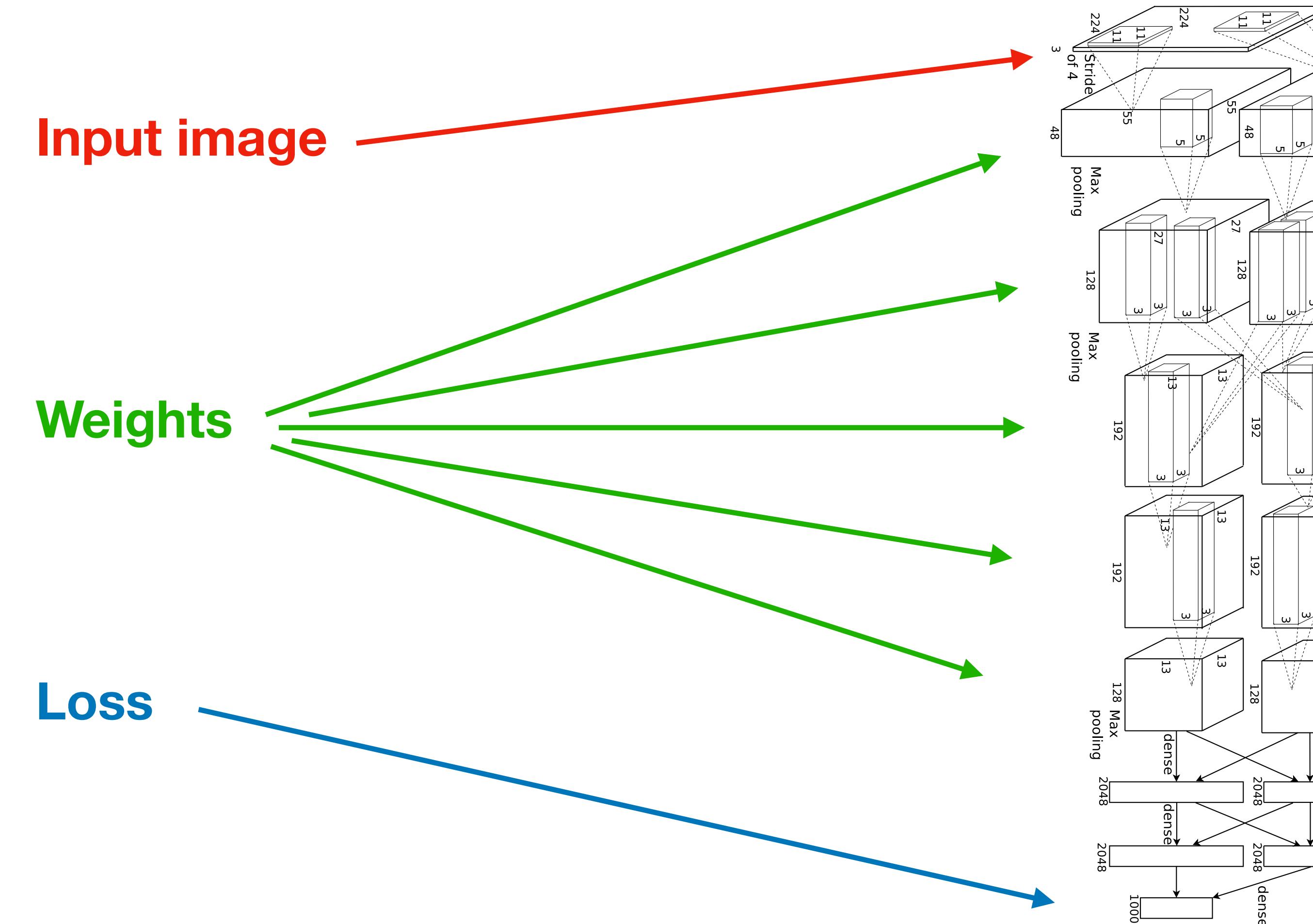
$$\nabla_W L = \nabla_W \left(\frac{1}{N} \sum_{i=1}^N \sum_{j \neq y_i} \max(0, W_{j,:} x - W_{y_i,:} x + 1) + \lambda \sum_k W_k^2 \right)$$



Better Idea: Computational Graphs

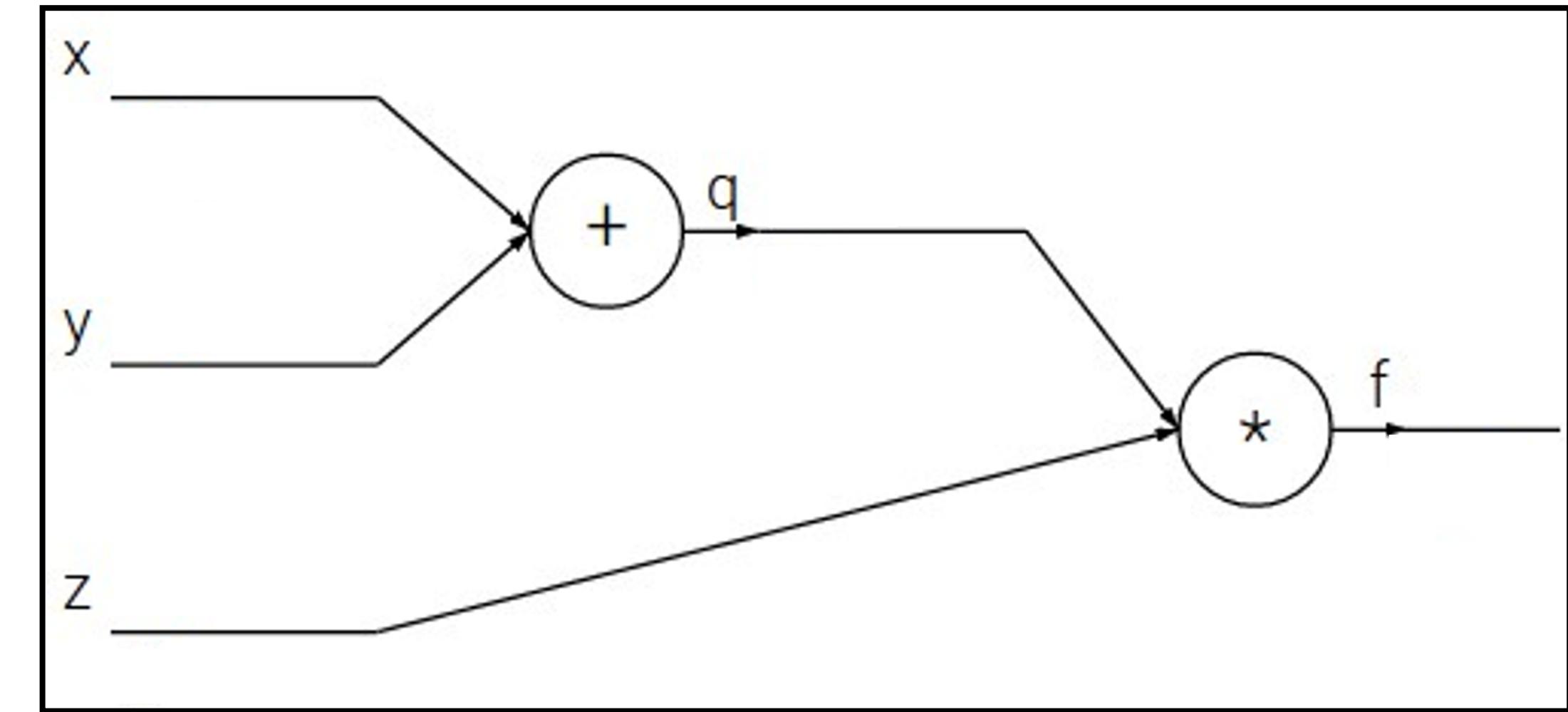


Deep Network (AlexNet)



Backpropagation: Simple Example

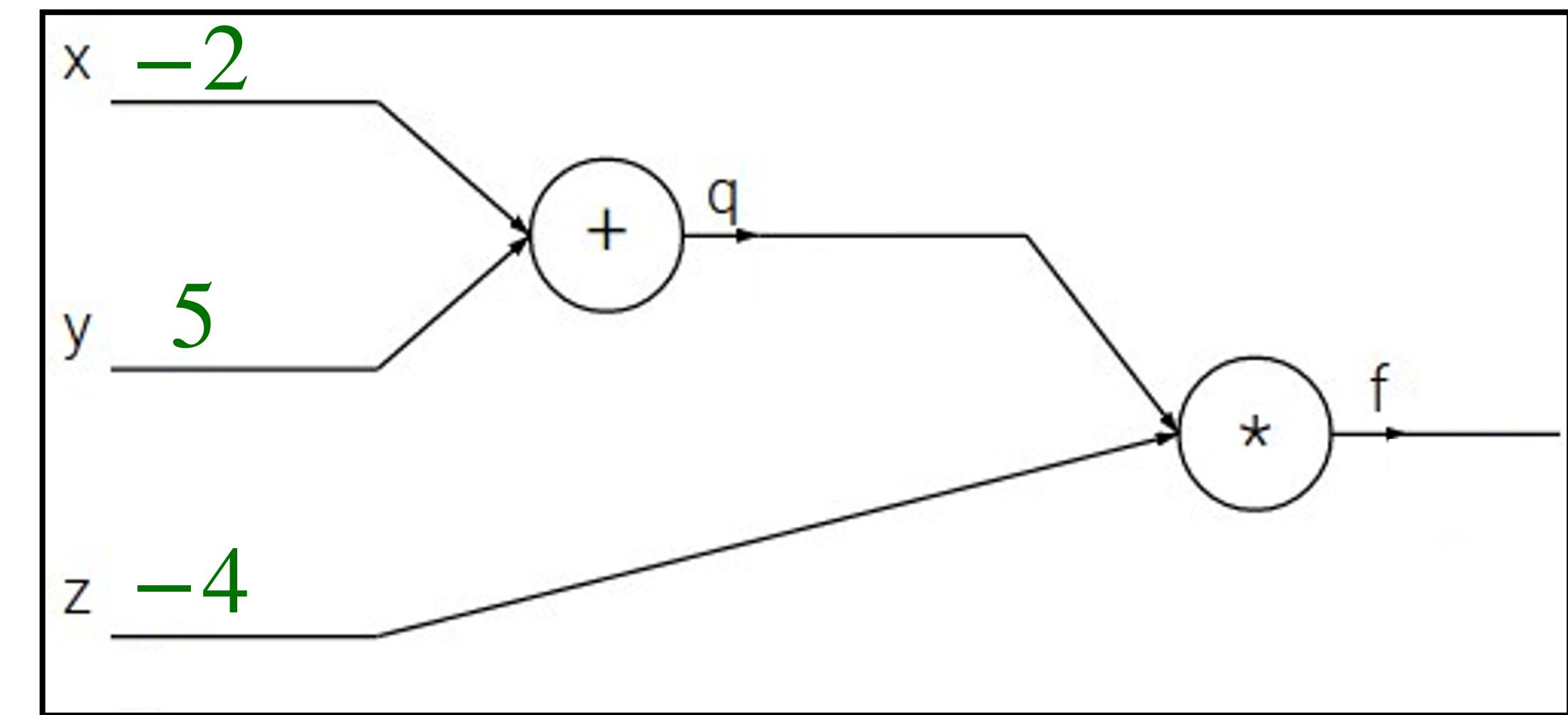
$$f(x, y, z) = (x + y) \cdot z$$



Backpropagation: Simple Example

$$f(x, y, z) = (x + y) \cdot z$$

e.g. $x = -2, y = 5, z = -4$



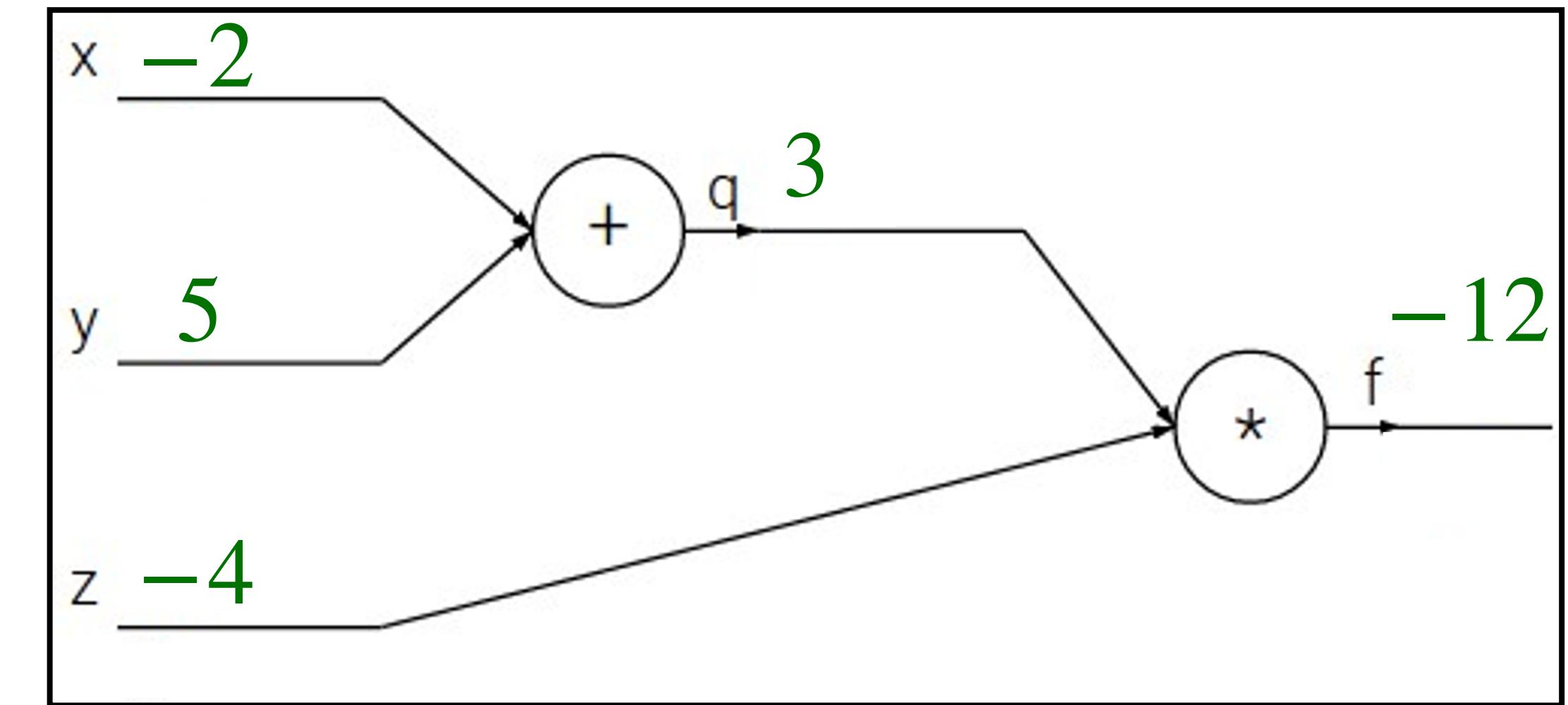
Backpropagation: Simple Example

$$f(x, y, z) = (x + y) \cdot z$$

e.g. $x = -2, y = 5, z = -4$

1. Forward pass: Compute outputs

$$q = x + y \quad f = q \cdot z$$



Backpropagation: Simple Example

$$f(x, y, z) = (x + y) \cdot z$$

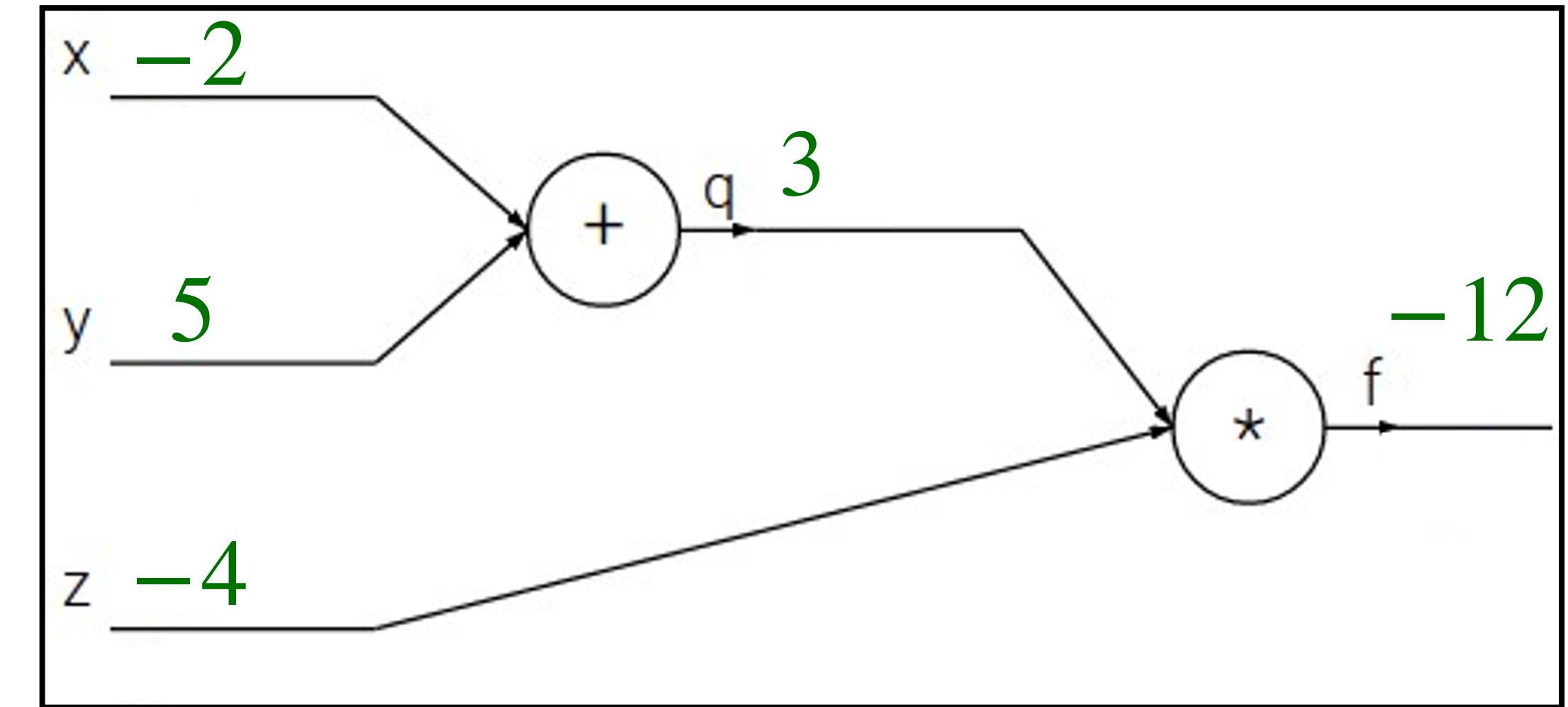
e.g. $x = -2, y = 5, z = -4$

1. Forward pass: Compute outputs

$$q = x + y \quad f = q \cdot z$$

2. Backward pass: Compute derivatives

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Backpropagation: Simple Example

$$f(x, y, z) = (x + y) \cdot z$$

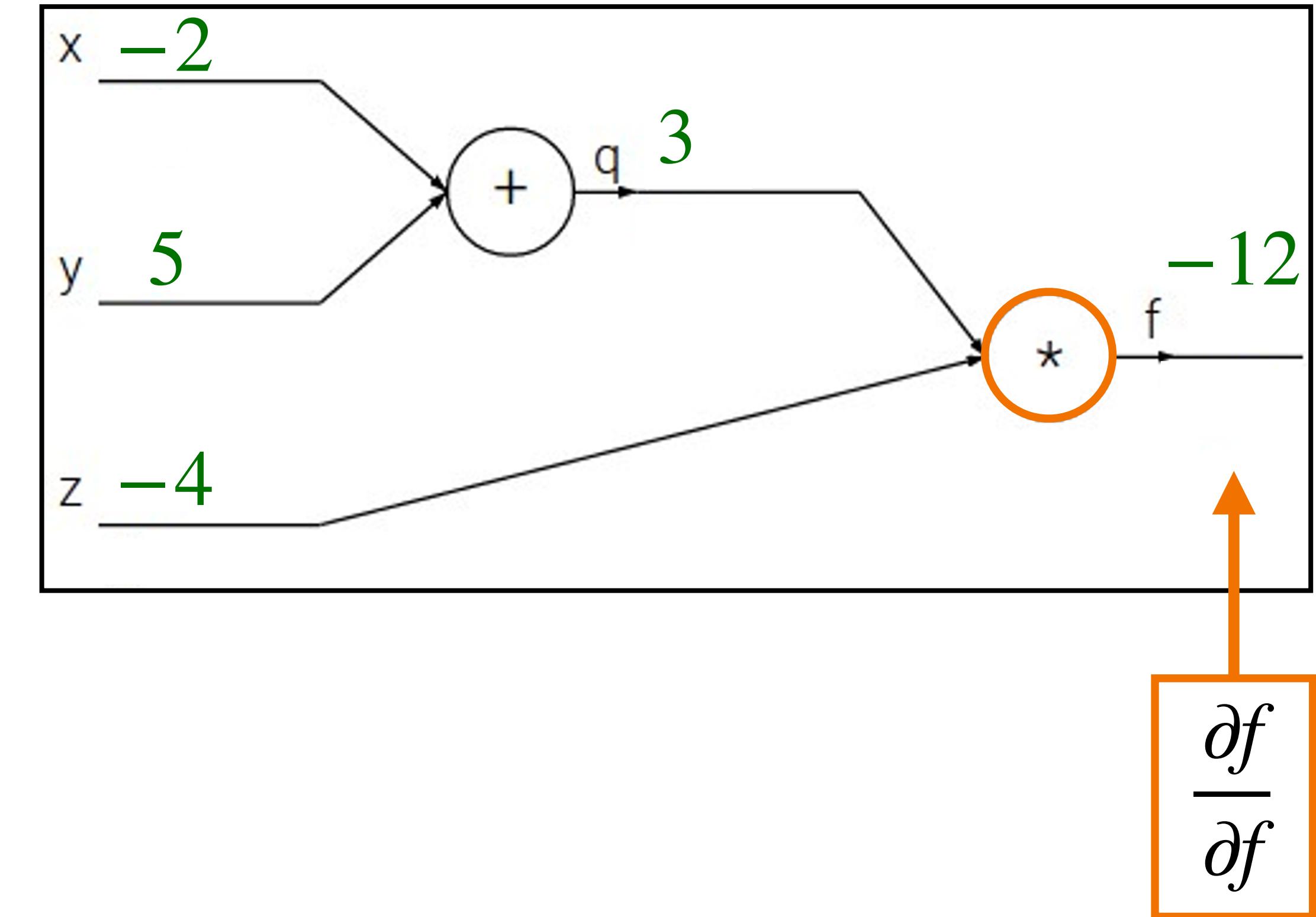
e.g. $x = -2, y = 5, z = -4$

1. Forward pass: Compute outputs

$$q = x + y \quad f = q \cdot z$$

2. Backward pass: Compute derivatives

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Backpropagation: Simple Example

$$f(x, y, z) = (x + y) \cdot z$$

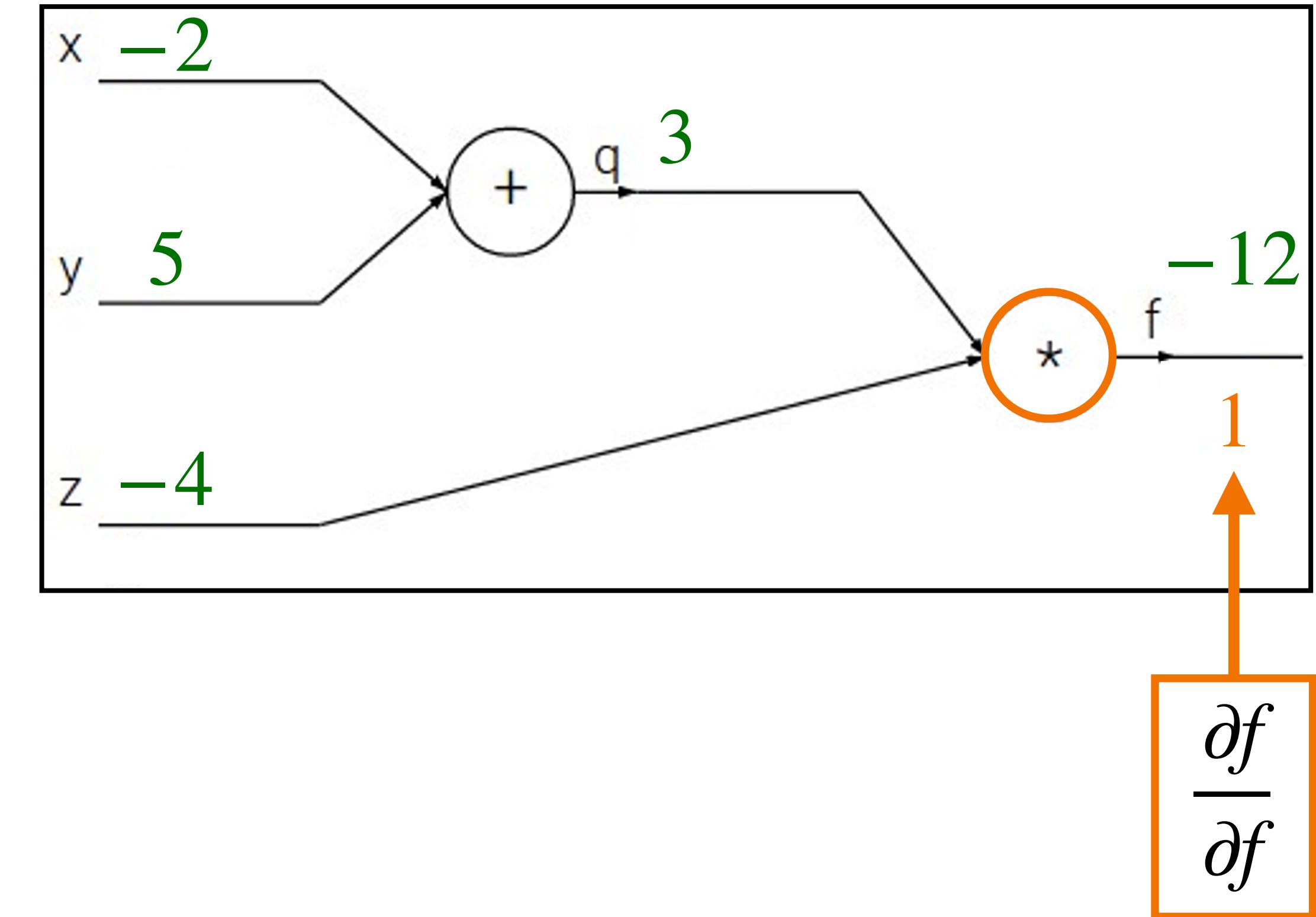
e.g. $x = -2, y = 5, z = -4$

1. Forward pass: Compute outputs

$$q = x + y \quad f = q \cdot z$$

2. Backward pass: Compute derivatives

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Backpropagation: Simple Example

$$f(x, y, z) = (x + y) \cdot z$$

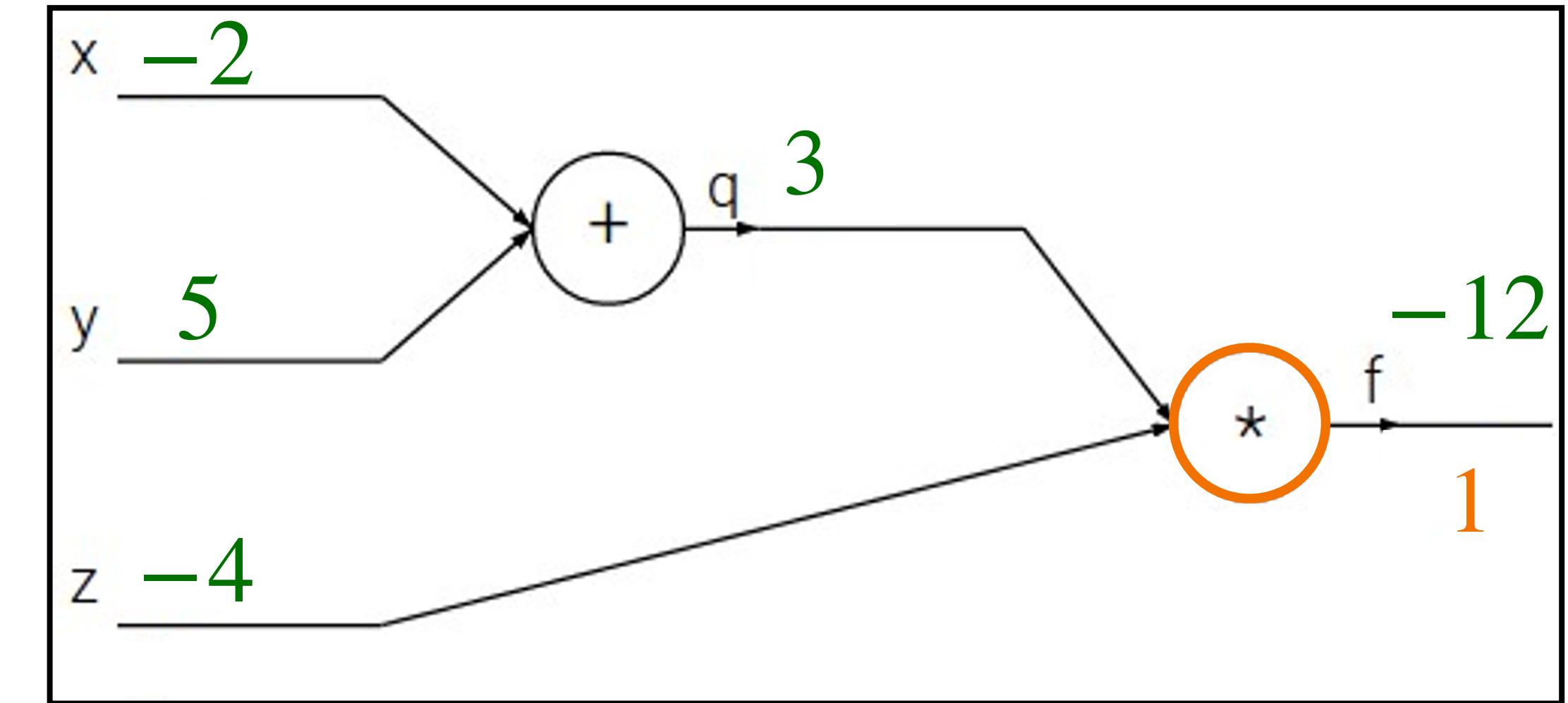
e.g. $x = -2, y = 5, z = -4$

1. Forward pass: Compute outputs

$$q = x + y \quad f = q \cdot z$$

2. Backward pass: Compute derivatives

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Backpropagation: Simple Example

$$f(x, y, z) = (x + y) \cdot z$$

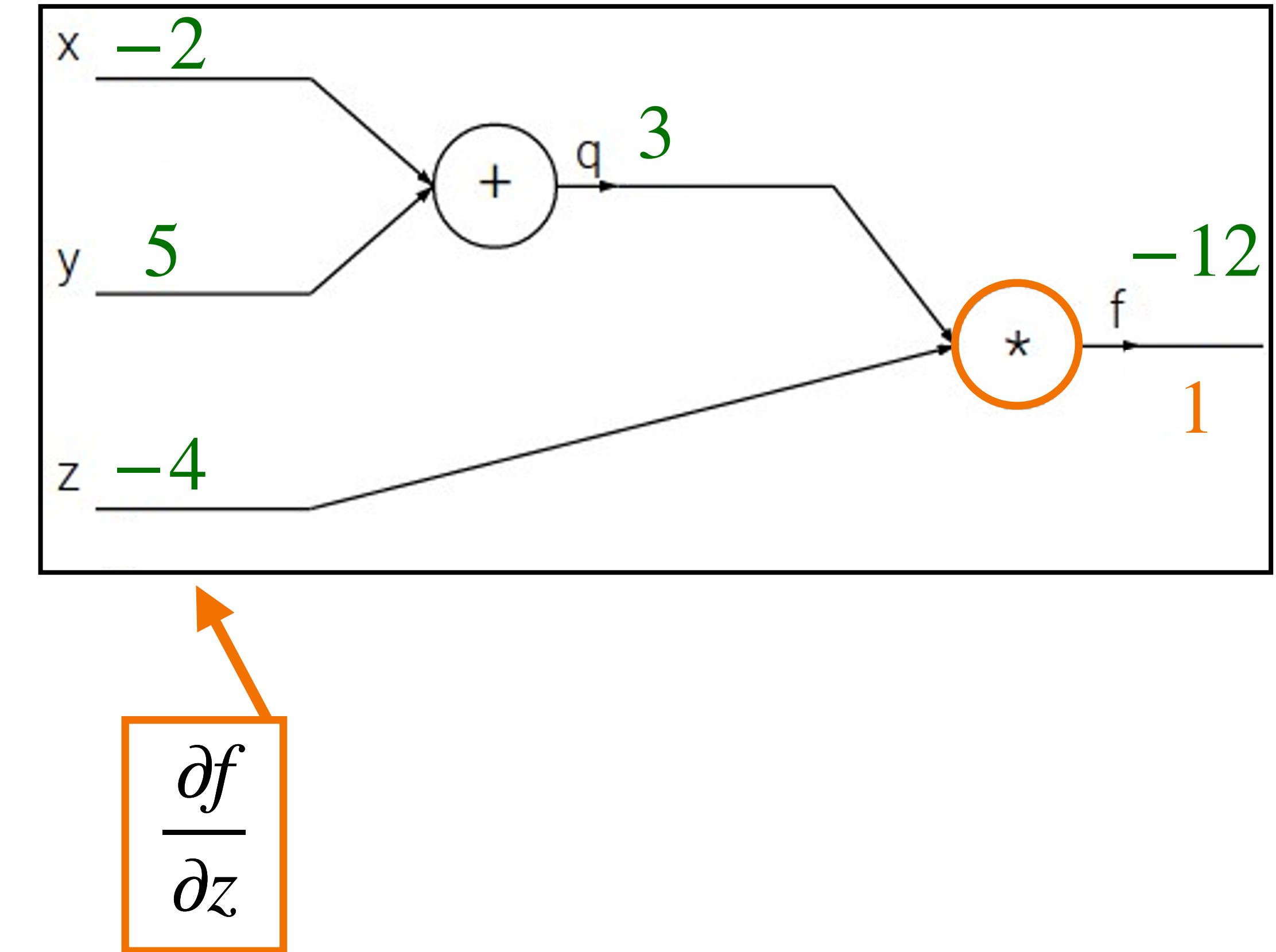
e.g. $x = -2, y = 5, z = -4$

1. Forward pass: Compute outputs

$$q = x + y \quad f = q \cdot z$$

2. Backward pass: Compute derivatives

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Backpropagation: Simple Example

$$f(x, y, z) = (x + y) \cdot z$$

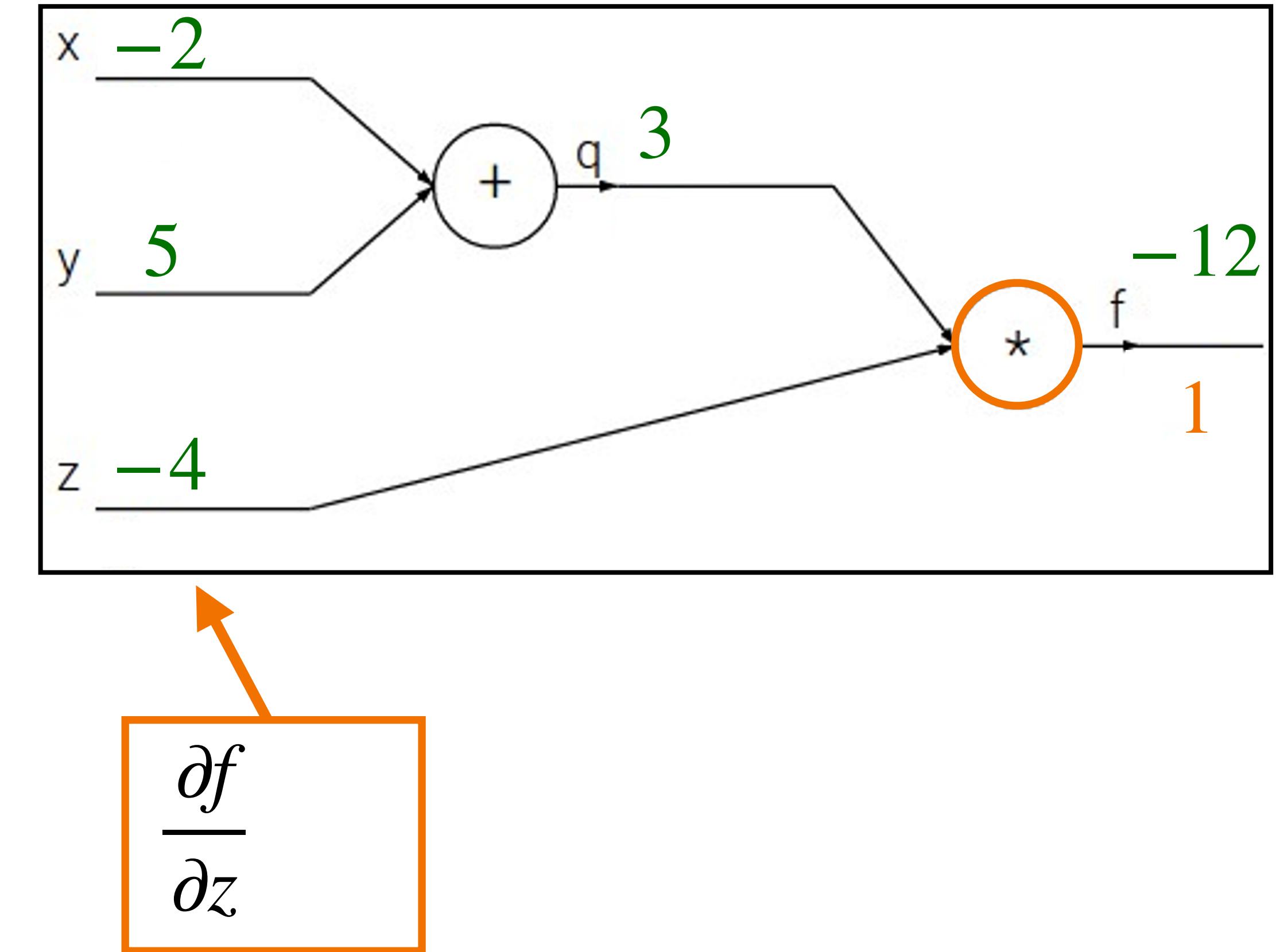
e.g. $x = -2, y = 5, z = -4$

1. Forward pass: Compute outputs

$$q = x + y \quad f = q \cdot z$$

2. Backward pass: Compute derivatives

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Backpropagation: Simple Example

$$f(x, y, z) = (x + y) \cdot z$$

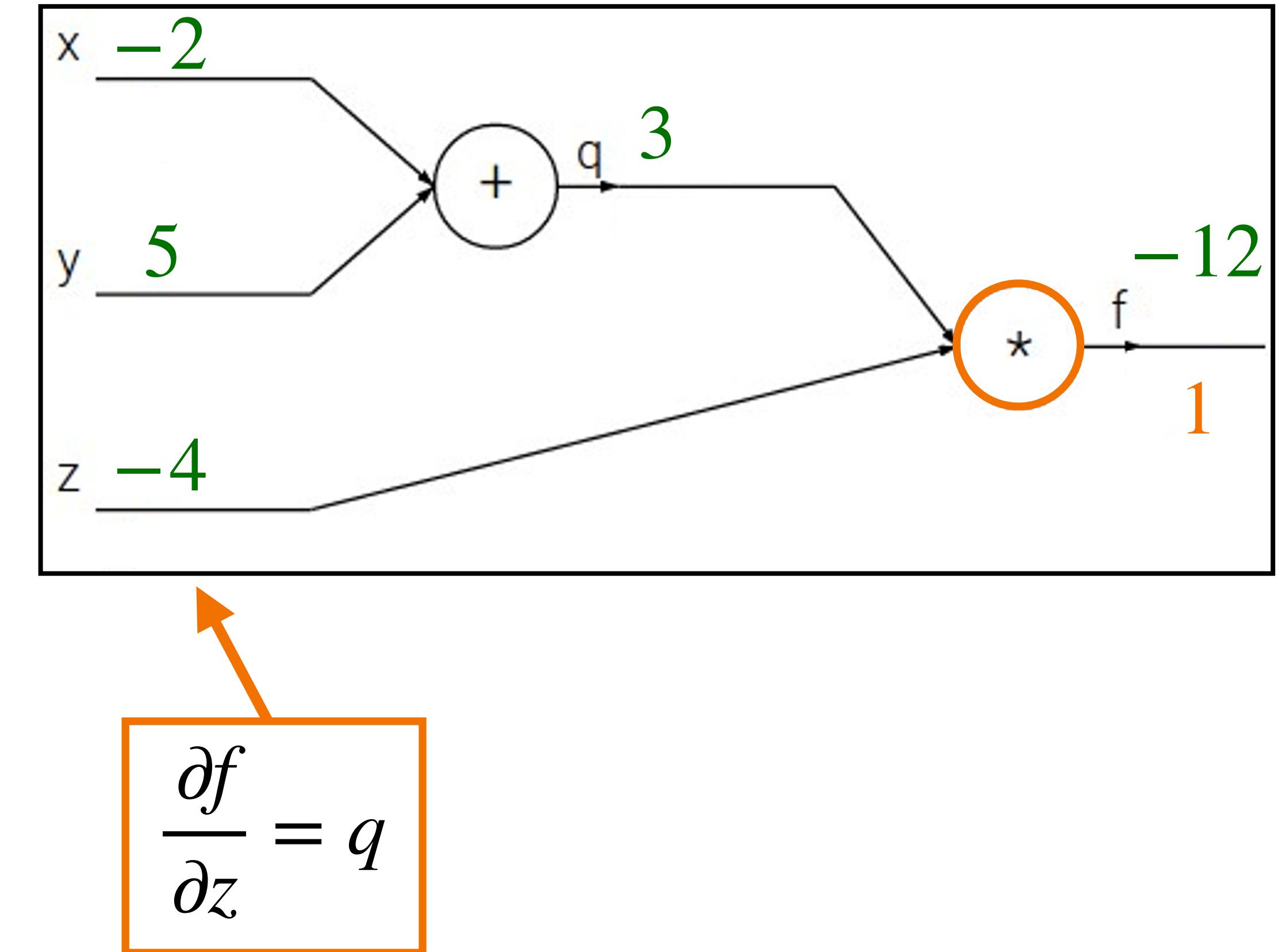
e.g. $x = -2, y = 5, z = -4$

1. Forward pass: Compute outputs

$$q = x + y \quad f = q \cdot z$$

2. Backward pass: Compute derivatives

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



$$\frac{\partial f}{\partial z} = q$$

Backpropagation: Simple Example

$$f(x, y, z) = (x + y) \cdot z$$

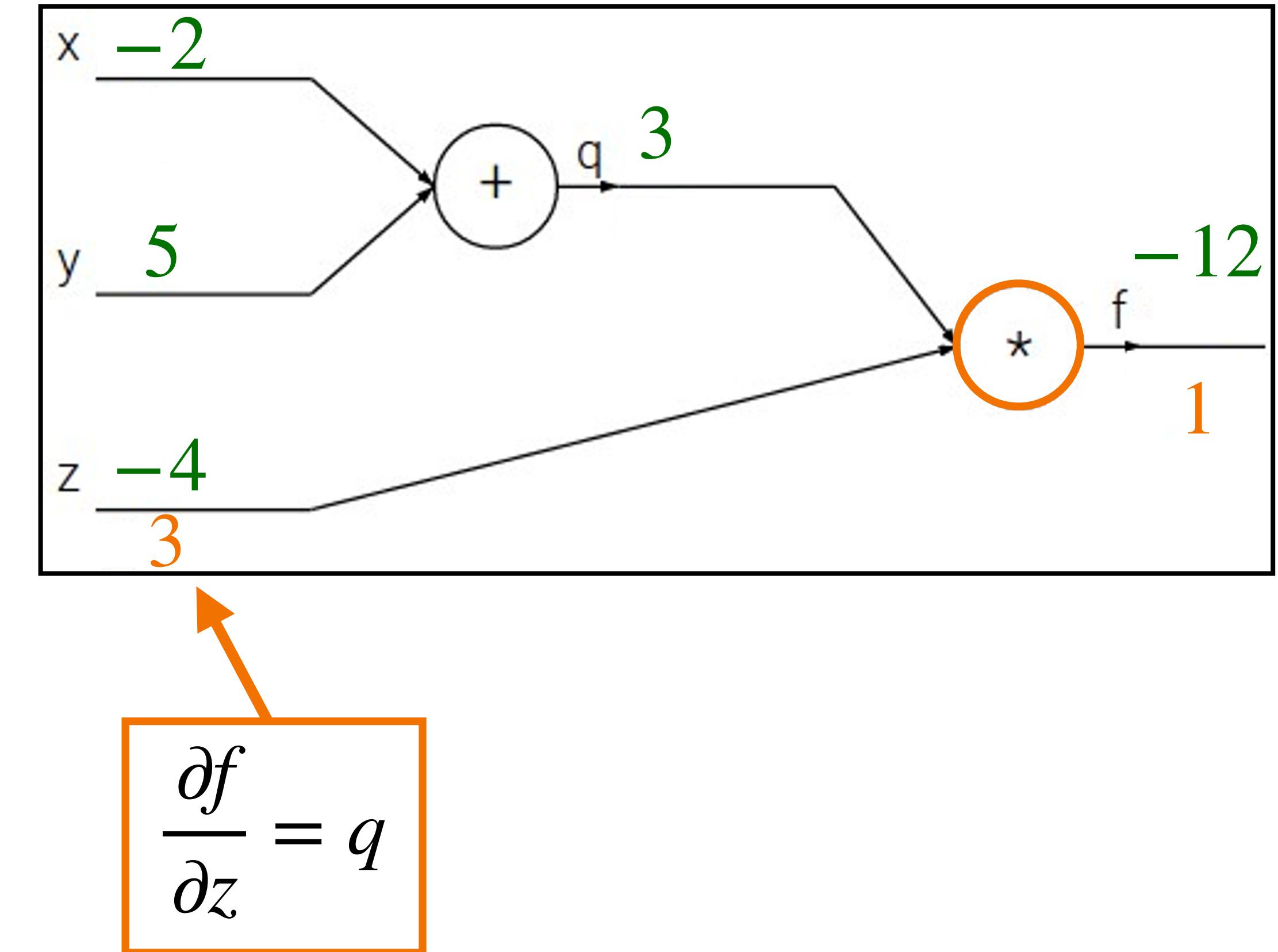
e.g. $x = -2, y = 5, z = -4$

1. Forward pass: Compute outputs

$$q = x + y \quad f = q \cdot z$$

2. Backward pass: Compute derivatives

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



$$\frac{\partial f}{\partial z} = q$$

Backpropagation: Simple Example

$$f(x, y, z) = (x + y) \cdot z$$

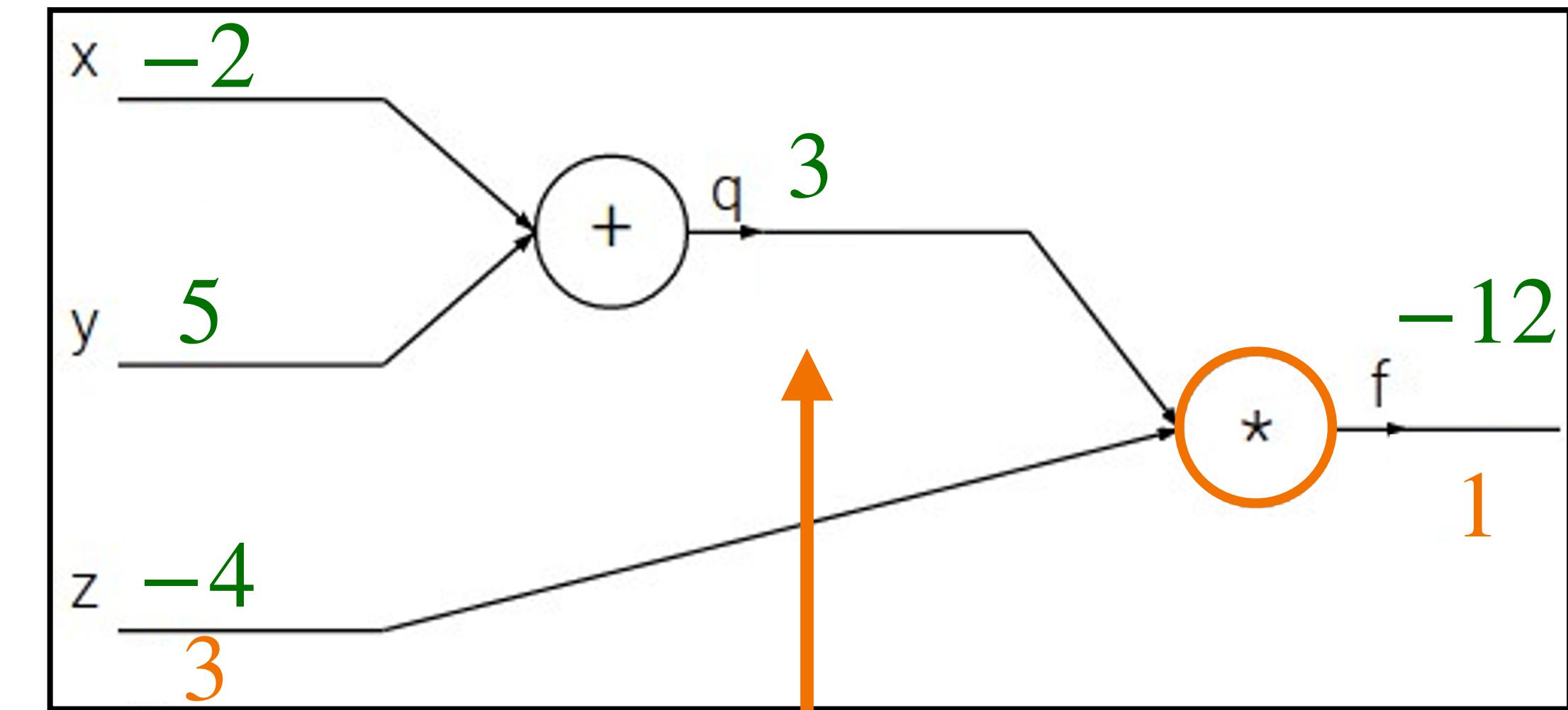
e.g. $x = -2, y = 5, z = -4$

1. Forward pass: Compute outputs

$$q = x + y \quad f = q \cdot z$$

2. Backward pass: Compute derivatives

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



$$\frac{\partial f}{\partial q}$$

Backpropagation: Simple Example

$$f(x, y, z) = (x + y) \cdot z$$

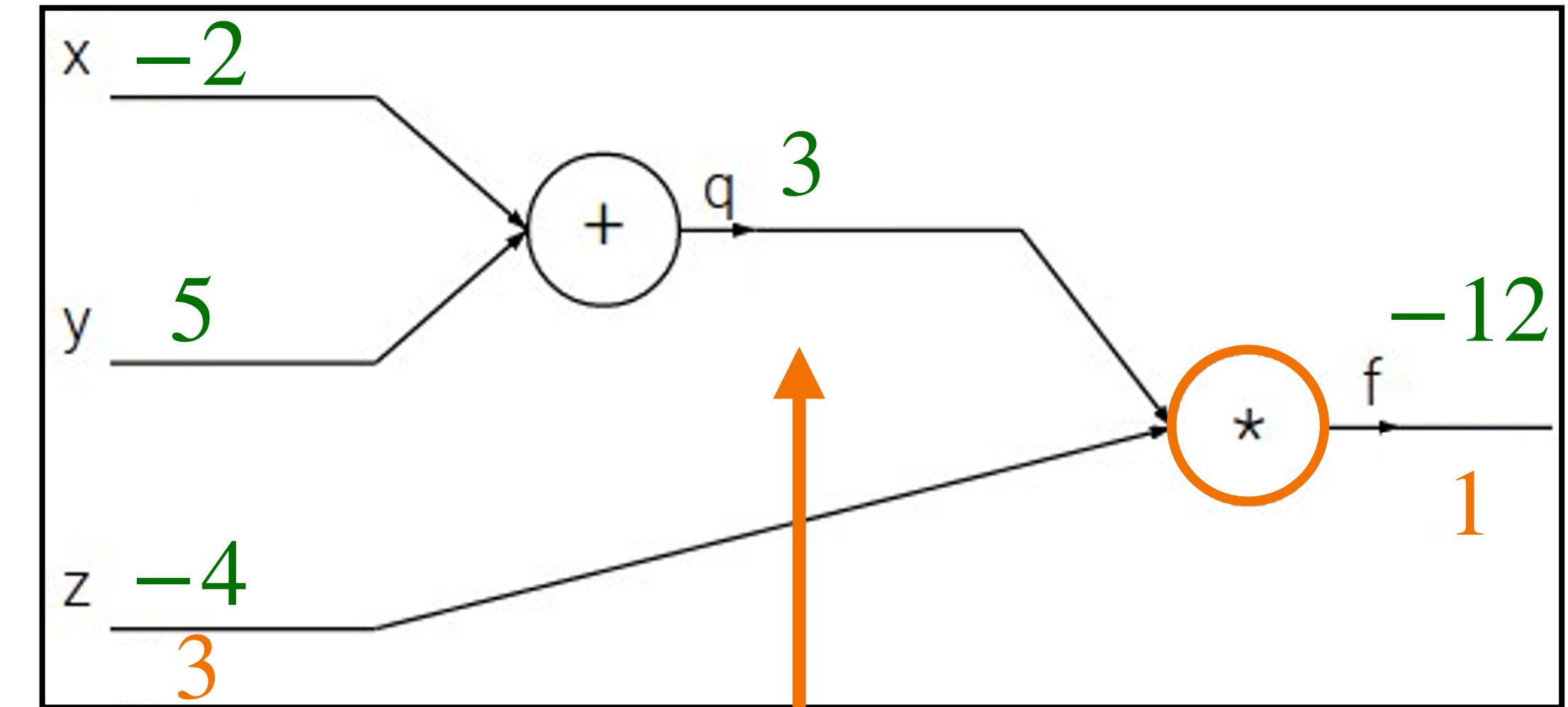
e.g. $x = -2, y = 5, z = -4$

1. Forward pass: Compute outputs

$$q = x + y \quad f = q \cdot z$$

2. Backward pass: Compute derivatives

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



$$\frac{\partial f}{\partial q}$$

Backpropagation: Simple Example

$$f(x, y, z) = (x + y) \cdot z$$

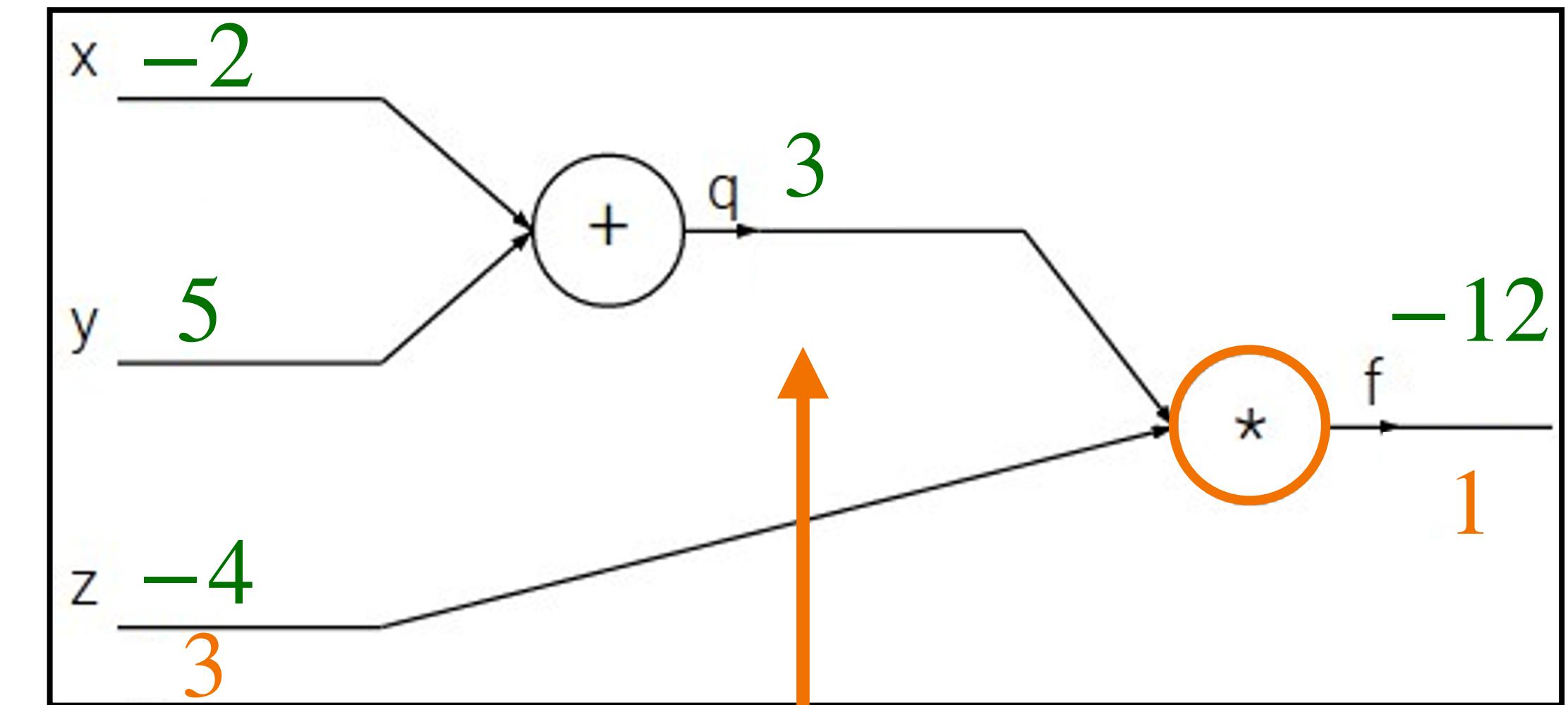
e.g. $x = -2, y = 5, z = -4$

1. Forward pass: Compute outputs

$$q = x + y \quad f = q \cdot z$$

2. Backward pass: Compute derivatives

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



$$\frac{\partial f}{\partial q} = z$$

Backpropagation: Simple Example

$$f(x, y, z) = (x + y) \cdot z$$

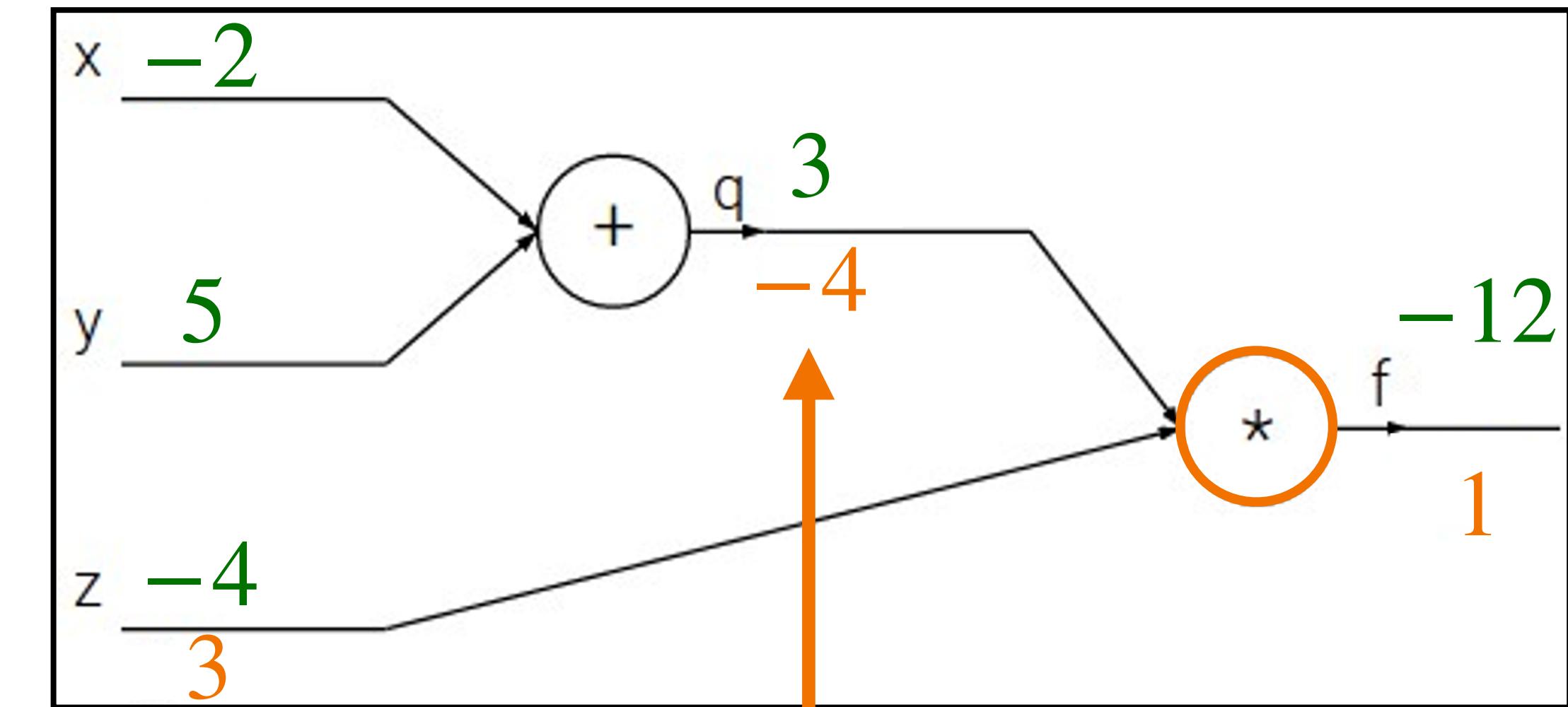
e.g. $x = -2, y = 5, z = -4$

1. Forward pass: Compute outputs

$$q = x + y \quad f = q \cdot z$$

2. Backward pass: Compute derivatives

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



$$\frac{\partial f}{\partial q} = z$$

Backpropagation: Simple Example

$$f(x, y, z) = (x + y) \cdot z$$

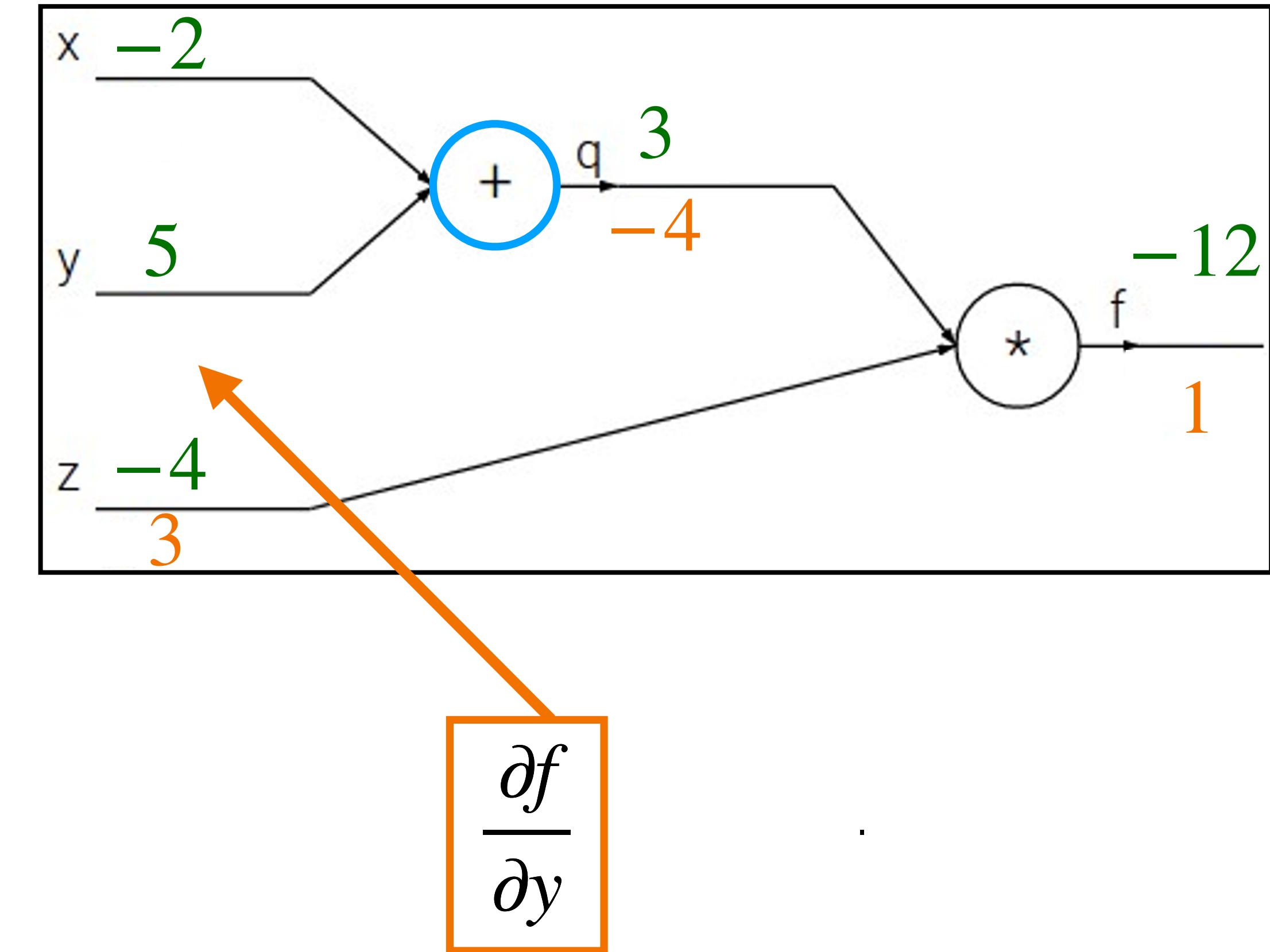
e.g. $x = -2, y = 5, z = -4$

1. Forward pass: Compute outputs

$$q = x + y \quad f = q \cdot z$$

2. Backward pass: Compute derivatives

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Backpropagation: Simple Example

$$f(x, y, z) = (x + y) \cdot z$$

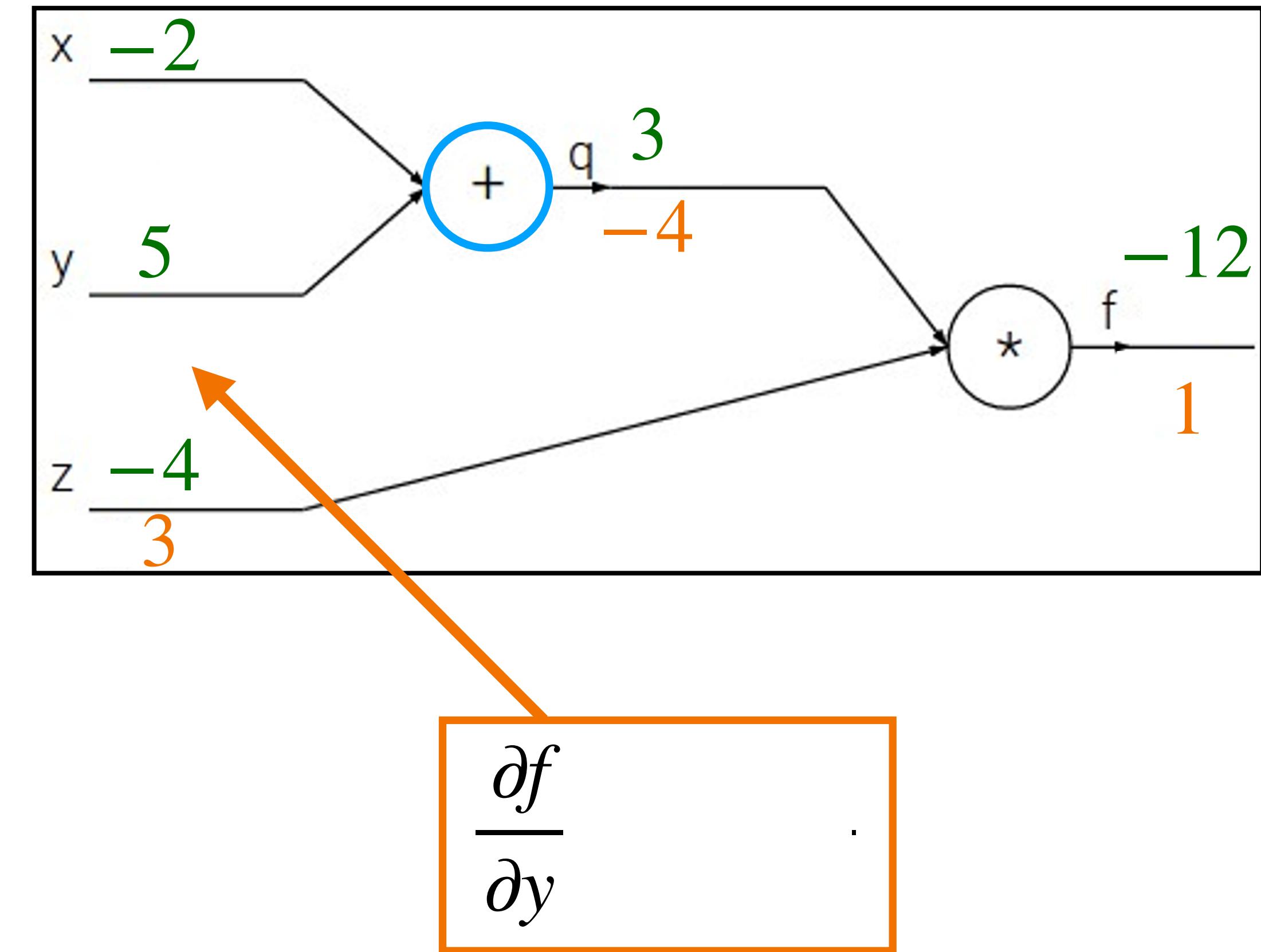
e.g. $x = -2, y = 5, z = -4$

1. Forward pass: Compute outputs

$$q = x + y \quad f = q \cdot z$$

2. Backward pass: Compute derivatives

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Backpropagation: Simple Example

$$f(x, y, z) = (x + y) \cdot z$$

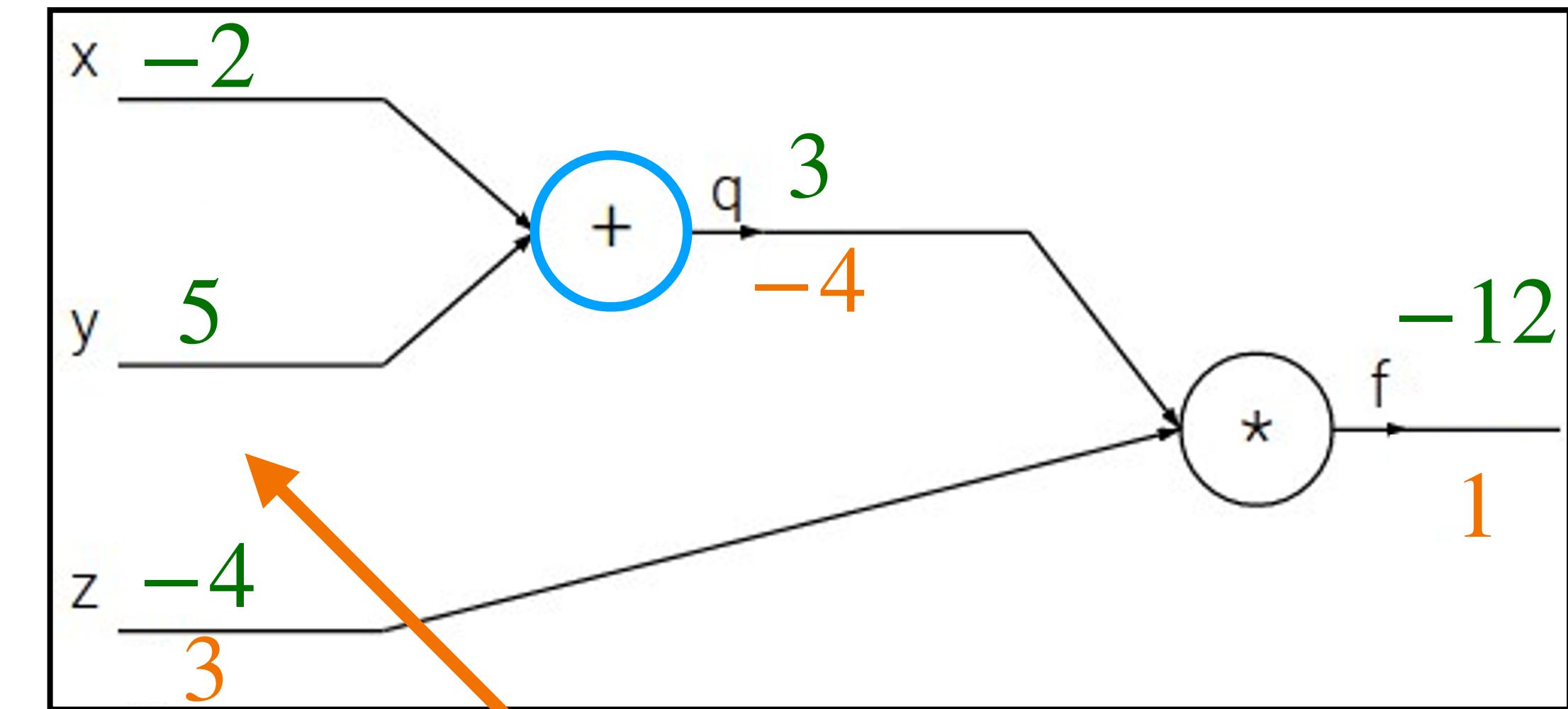
e.g. $x = -2, y = 5, z = -4$

1. Forward pass: Compute outputs

$$q = x + y \quad f = q \cdot z$$

2. Backward pass: Compute derivatives

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



$$\frac{\partial f}{\partial y} = \frac{\partial q}{\partial y} \frac{\partial f}{\partial q}$$

Backpropagation: Simple Example

$$f(x, y, z) = (x + y) \cdot z$$

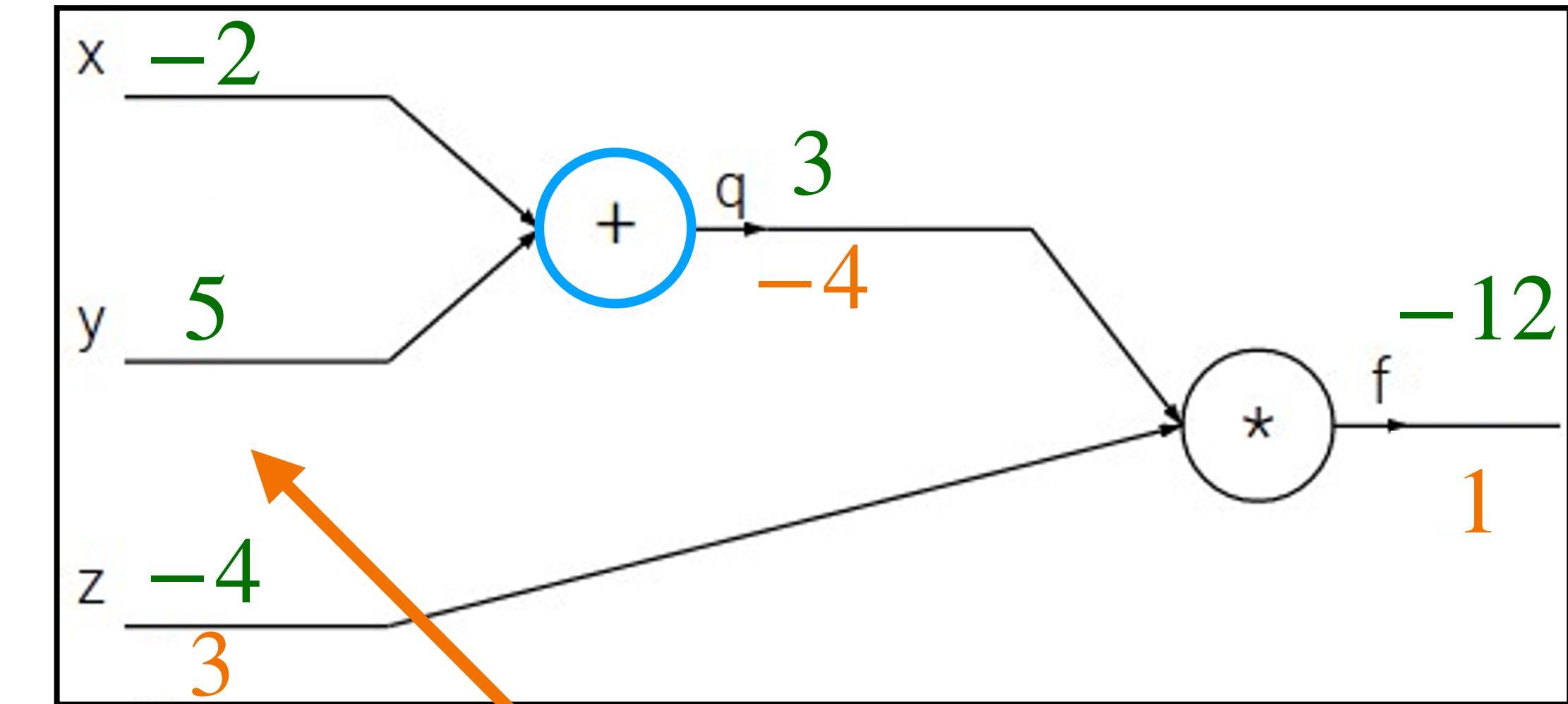
e.g. $x = -2, y = 5, z = -4$

1. Forward pass: Compute outputs

$$q = x + y \quad f = q \cdot z$$

2. Backward pass: Compute derivatives

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



$$\frac{\partial f}{\partial y} = \frac{\partial q}{\partial y} \frac{\partial f}{\partial q}$$

Downstream
Gradient

Local
Gradient

Upstream
Gradient

Backpropagation: Simple Example

$$f(x, y, z) = (x + y) \cdot z$$

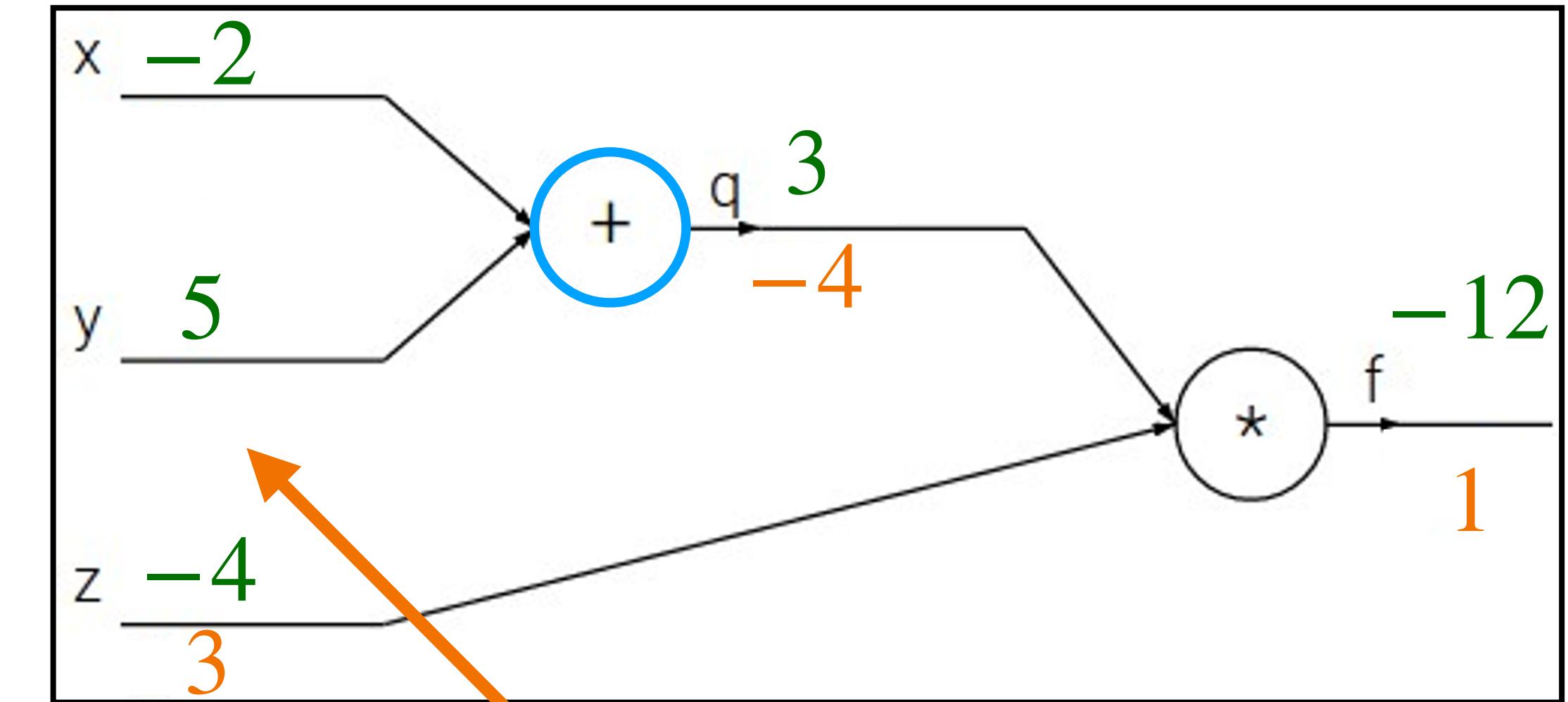
e.g. $x = -2, y = 5, z = -4$

1. Forward pass: Compute outputs

$$q = x + y \quad f = q \cdot z$$

2. Backward pass: Compute derivatives

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



$$\frac{\partial f}{\partial y} = \frac{\partial q}{\partial y} \frac{\partial f}{\partial q}$$

$$\frac{\partial q}{\partial y} = 1$$

Downstream
Gradient

Local
Gradient

Upstream
Gradient

Backpropagation: Simple Example

$$f(x, y, z) = (x + y) \cdot z$$

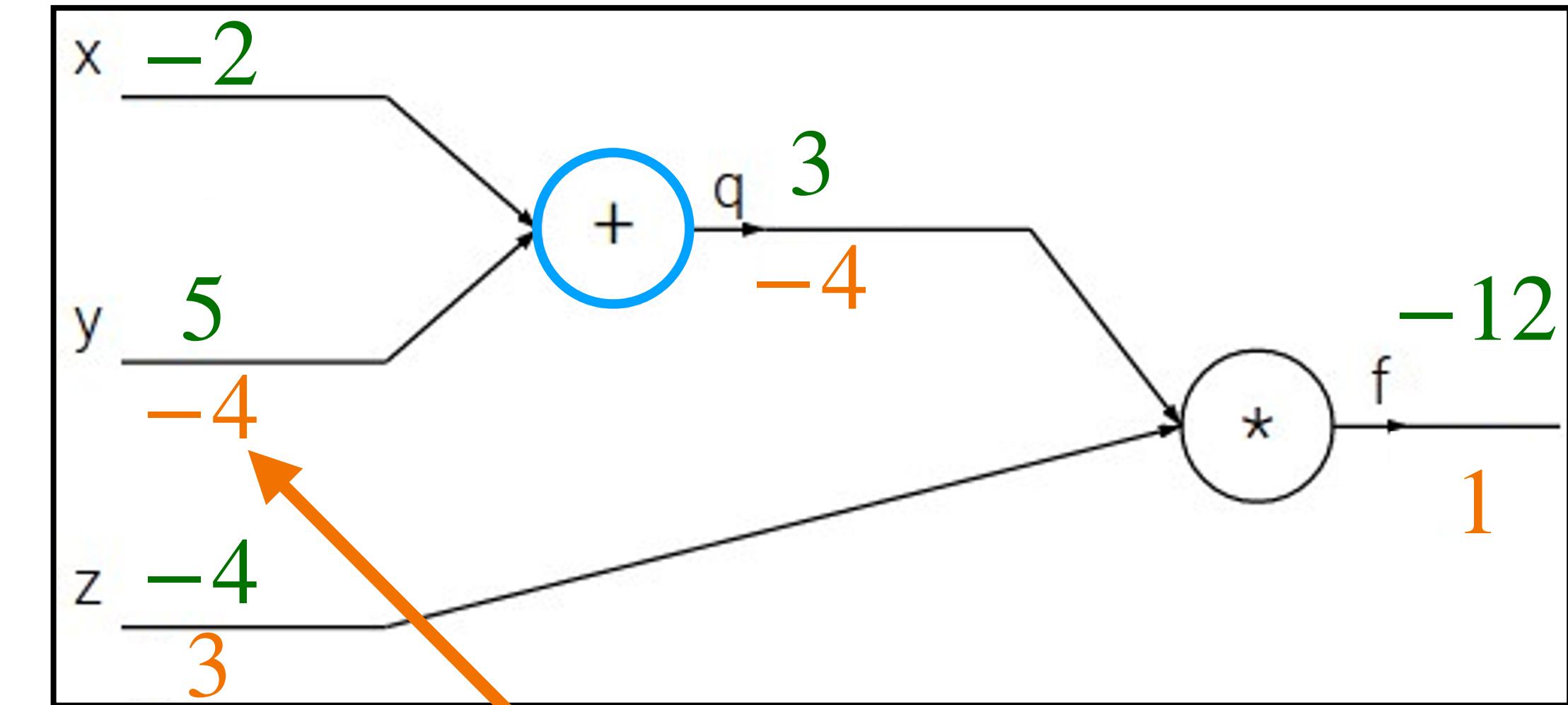
e.g. $x = -2, y = 5, z = -4$

1. Forward pass: Compute outputs

$$q = x + y \quad f = q \cdot z$$

2. Backward pass: Compute derivatives

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



$$\frac{\partial f}{\partial y} = \frac{\partial q}{\partial y} \frac{\partial f}{\partial q}$$

$$\frac{\partial q}{\partial y} = 1$$

Downstream
Gradient

Local
Gradient

Upstream
Gradient

Backpropagation: Simple Example

$$f(x, y, z) = (x + y) \cdot z$$

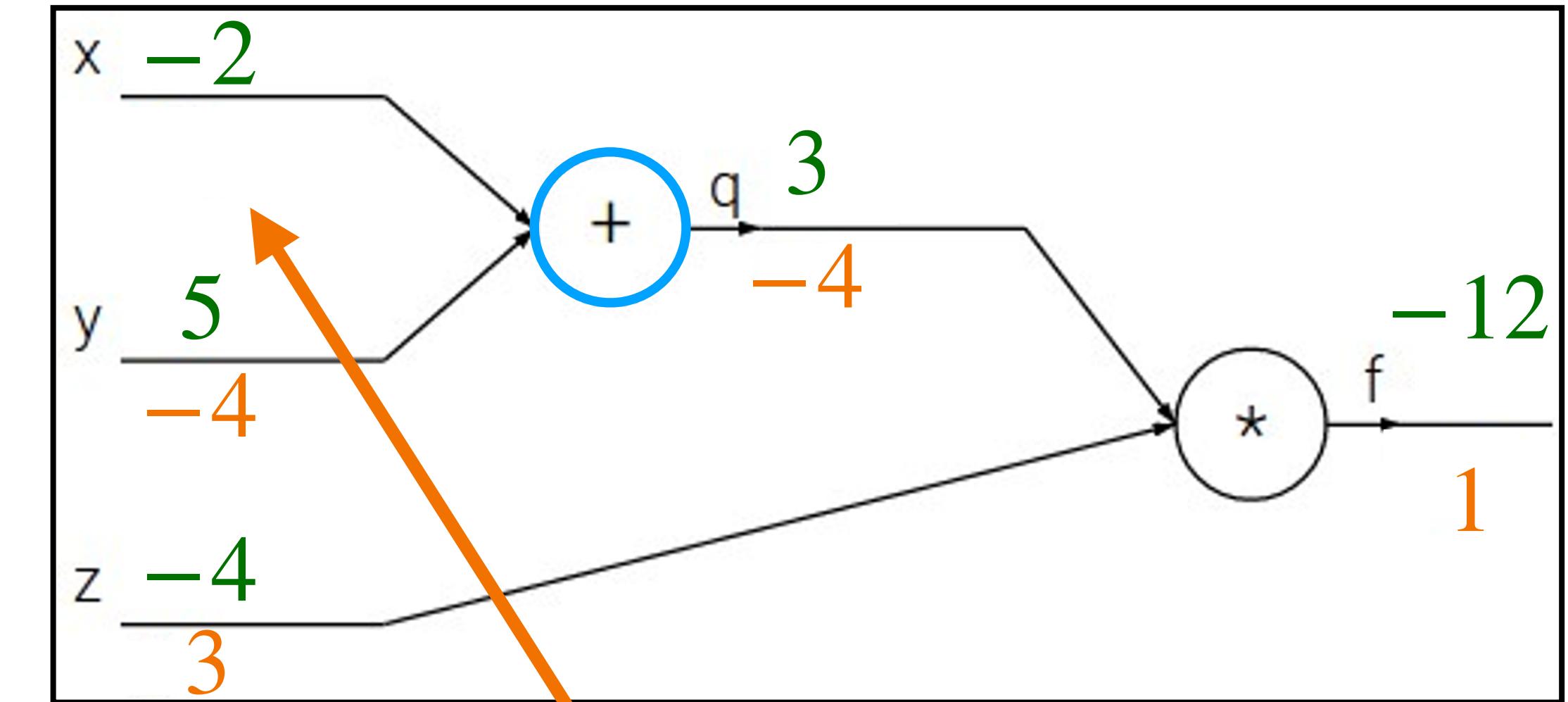
e.g. $x = -2, y = 5, z = -4$

1. Forward pass: Compute outputs

$$q = x + y \quad f = q \cdot z$$

2. Backward pass: Compute derivatives

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



$$\frac{\partial f}{\partial x} = \frac{\partial q}{\partial x} \frac{\partial f}{\partial q}$$

$$\frac{\partial q}{\partial x} = 1$$

Downstream
Gradient

Local
Gradient

Upstream
Gradient

Backpropagation: Simple Example

$$f(x, y, z) = (x + y) \cdot z$$

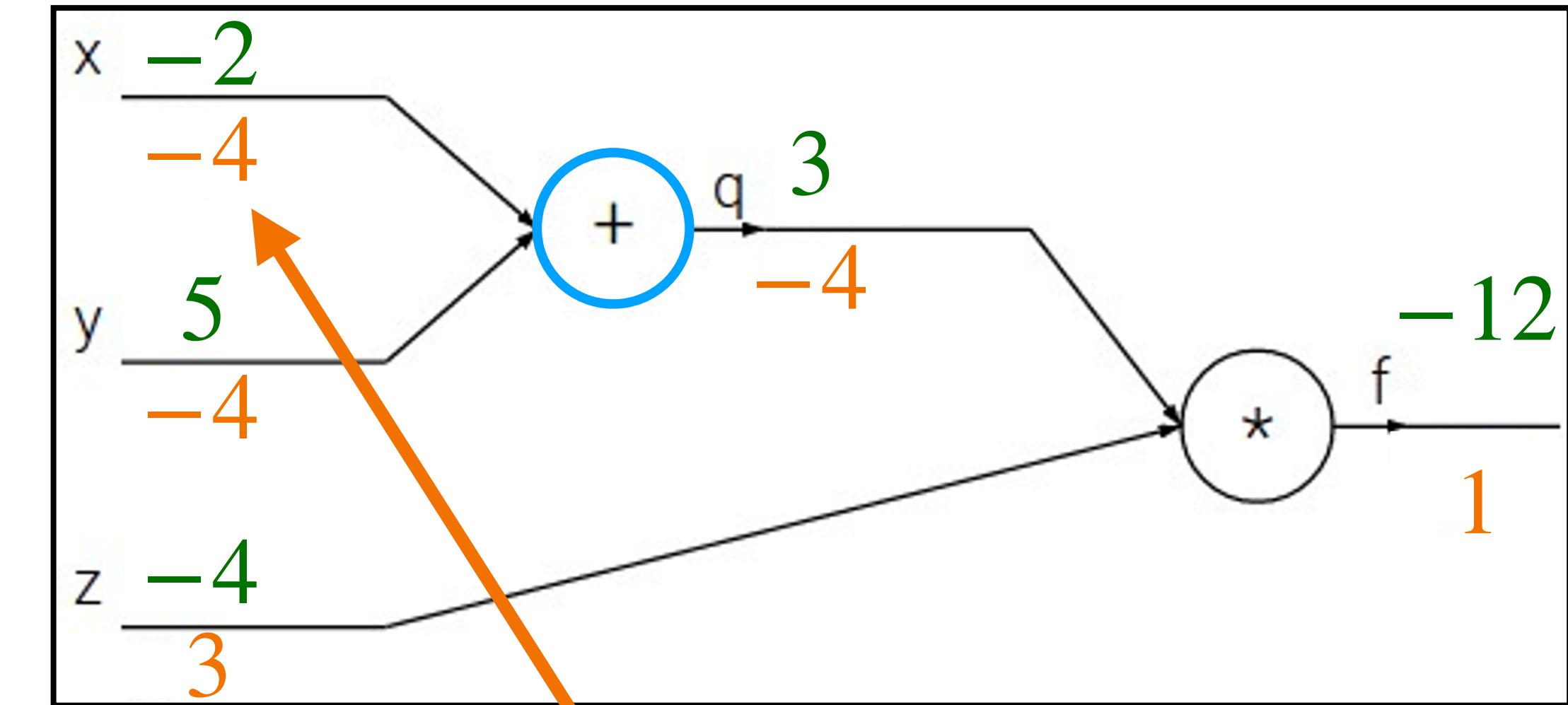
e.g. $x = -2, y = 5, z = -4$

1. Forward pass: Compute outputs

$$q = x + y \quad f = q \cdot z$$

2. Backward pass: Compute derivatives

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



$$\frac{\partial f}{\partial x} = \frac{\partial q}{\partial x} \frac{\partial f}{\partial q}$$

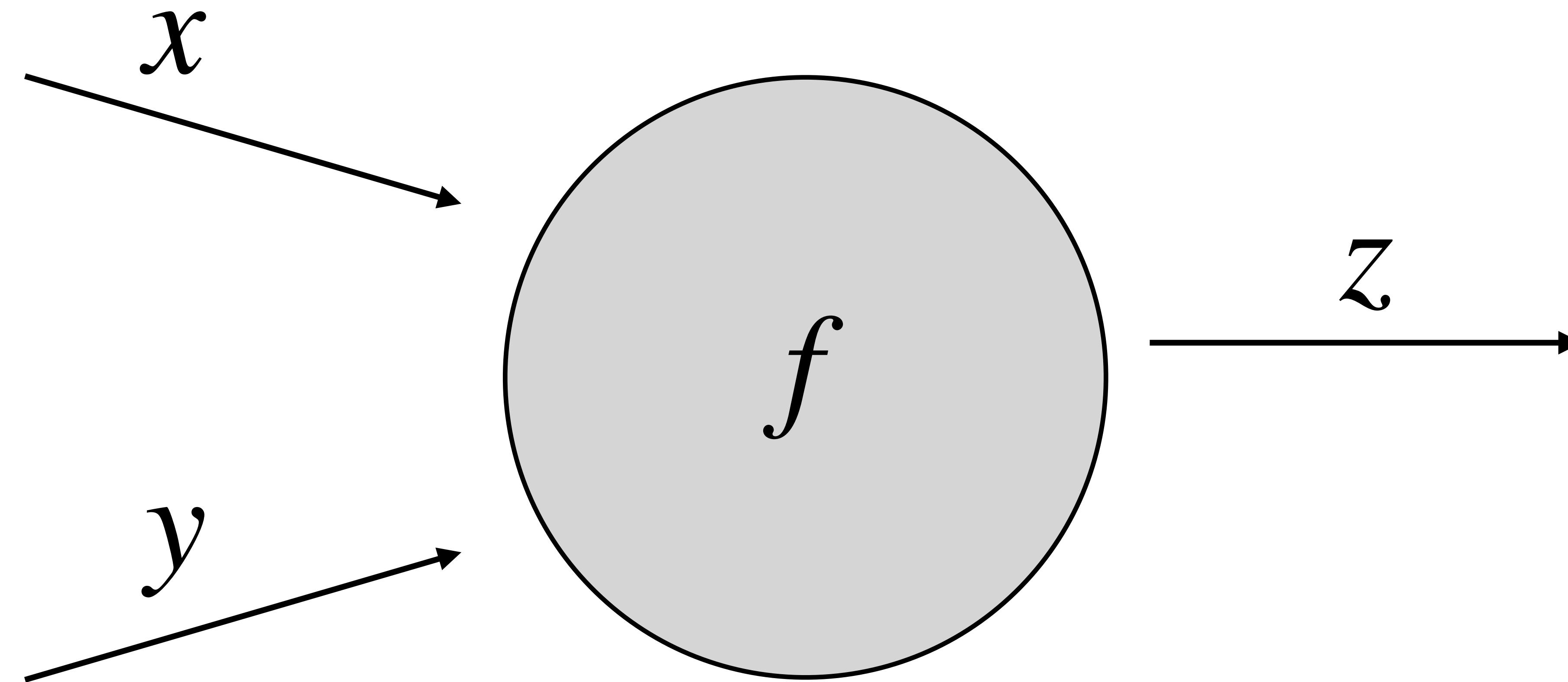
$$\frac{\partial q}{\partial x} = 1$$

Downstream
Gradient

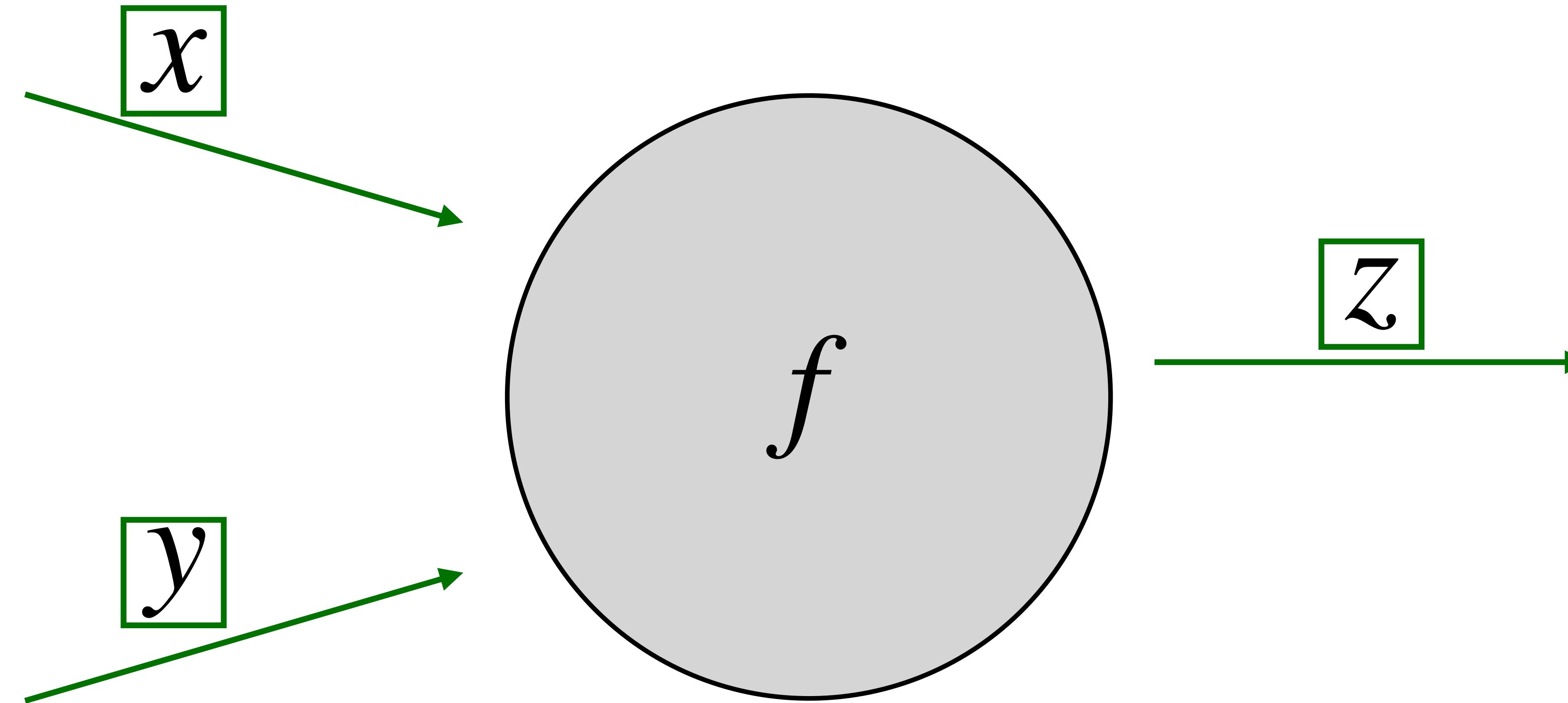
Local
Gradient

Upstream
Gradient

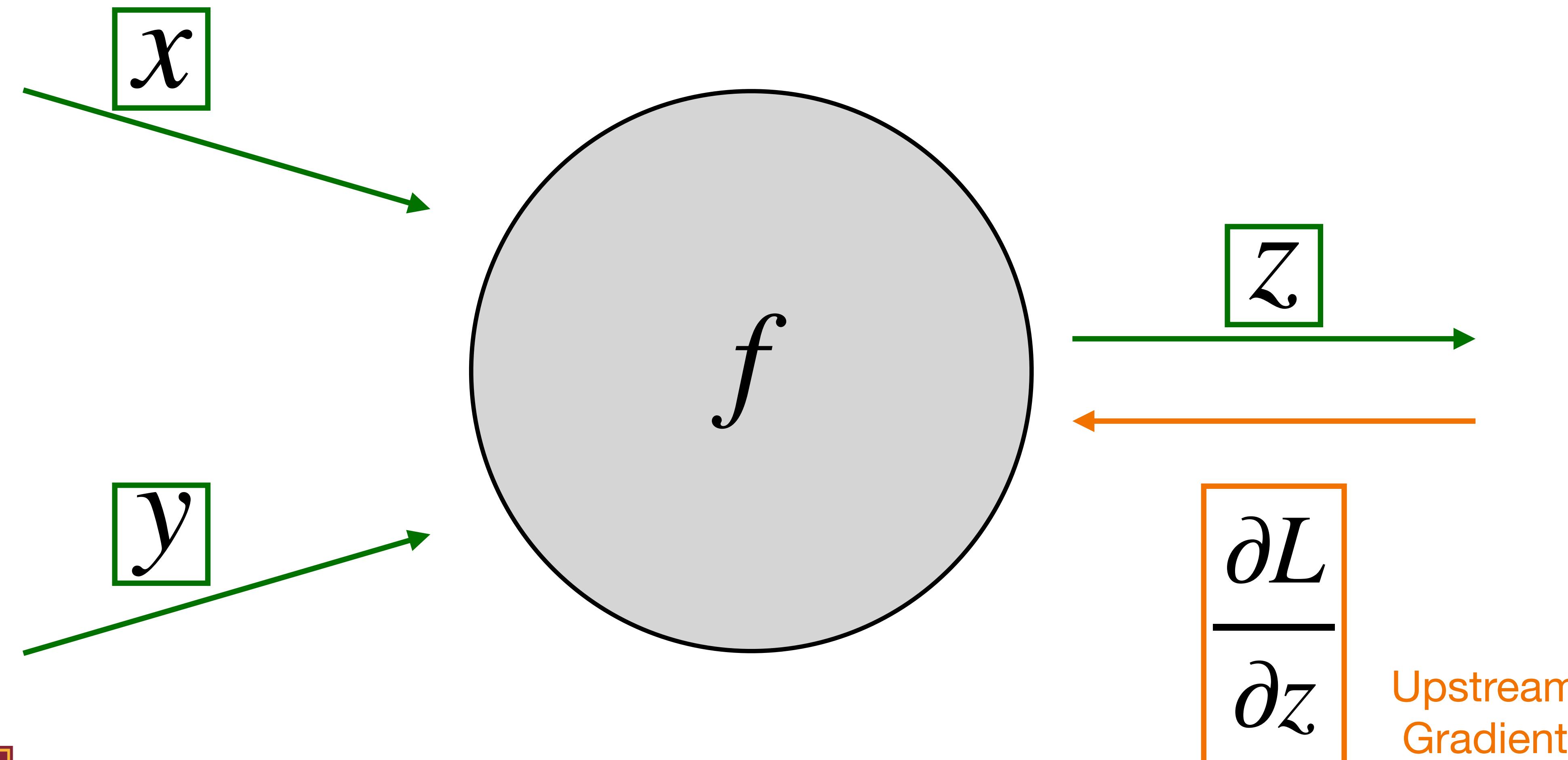
Generalizing Backpropagation



Generalizing Backpropagation

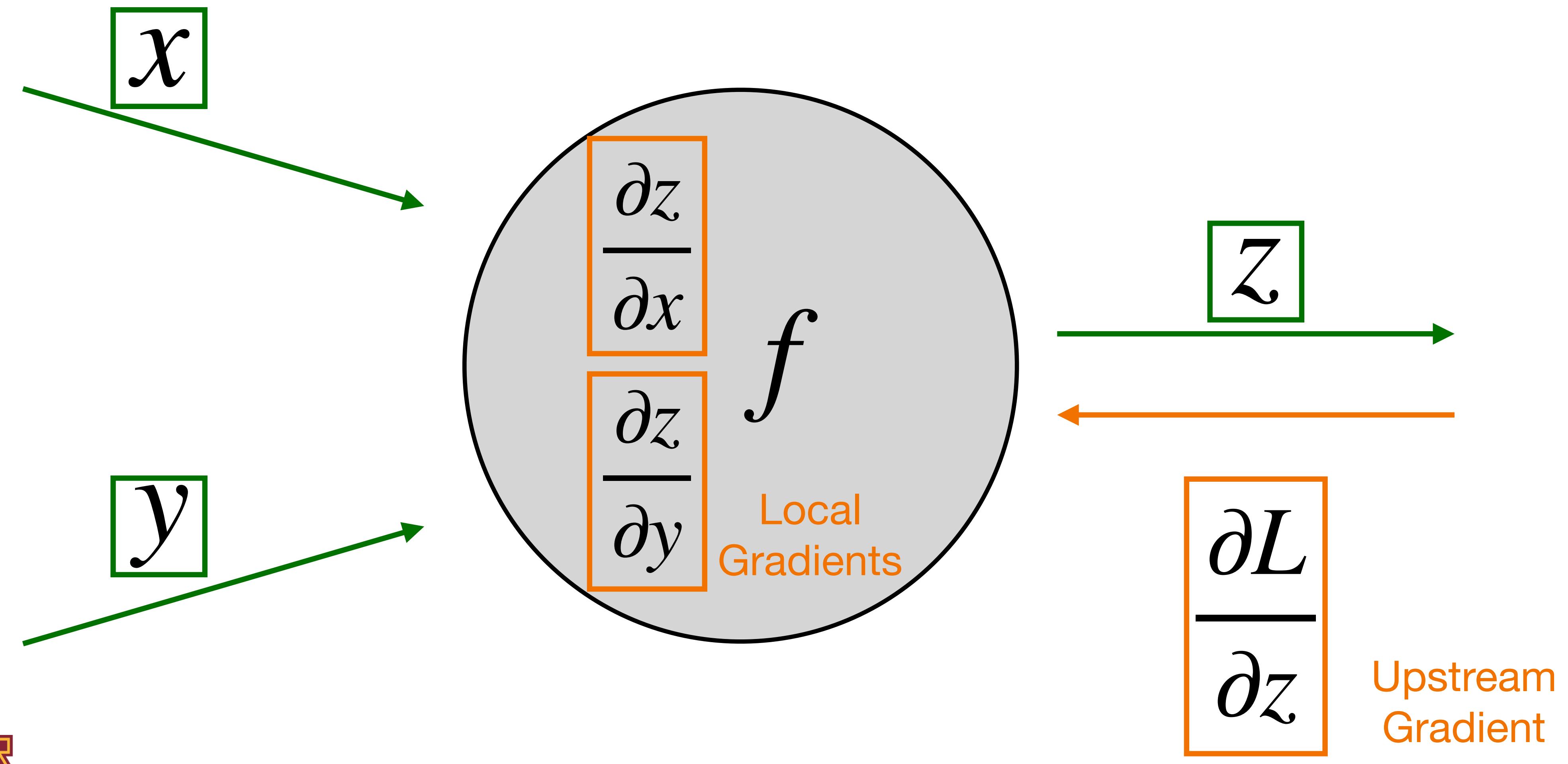


Generalizing Backpropagation

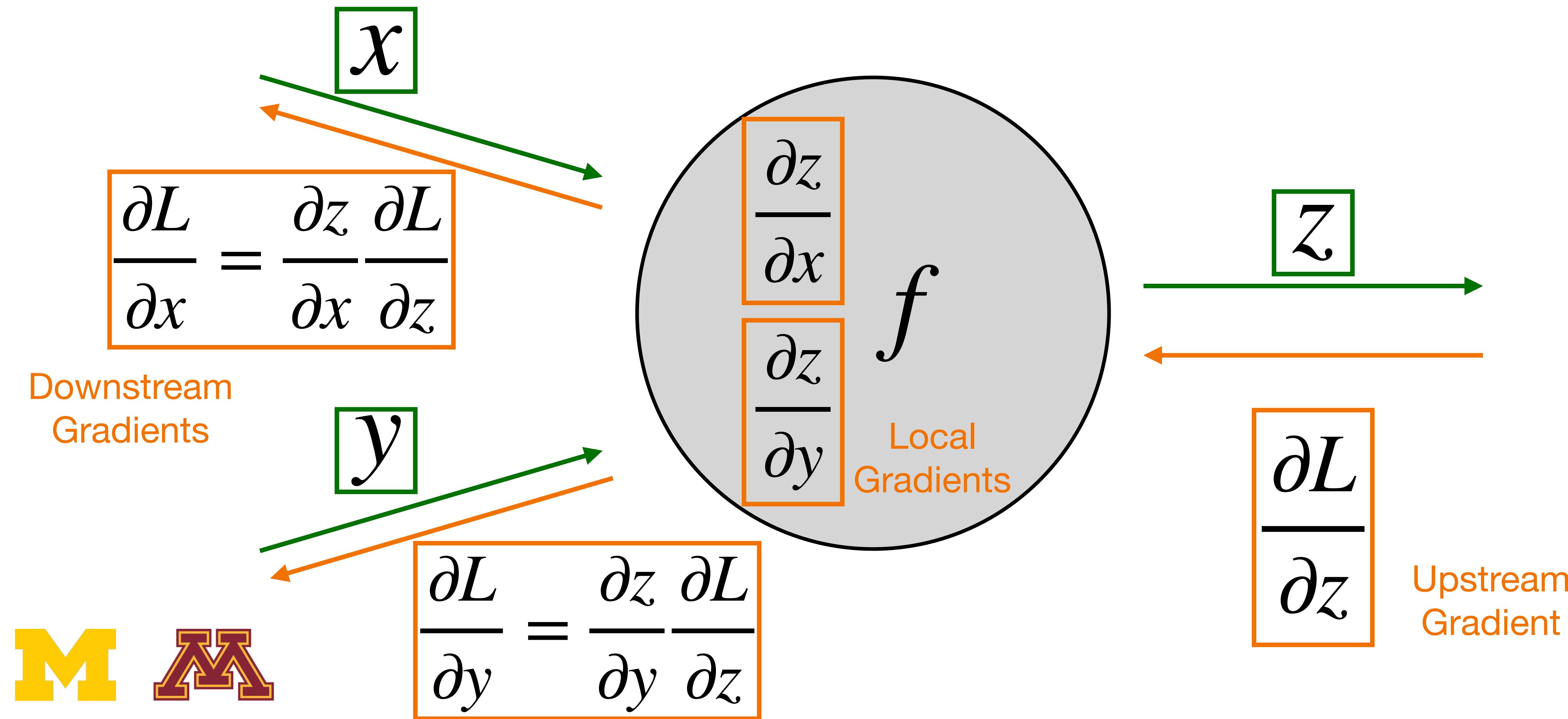


Upstream
Gradient

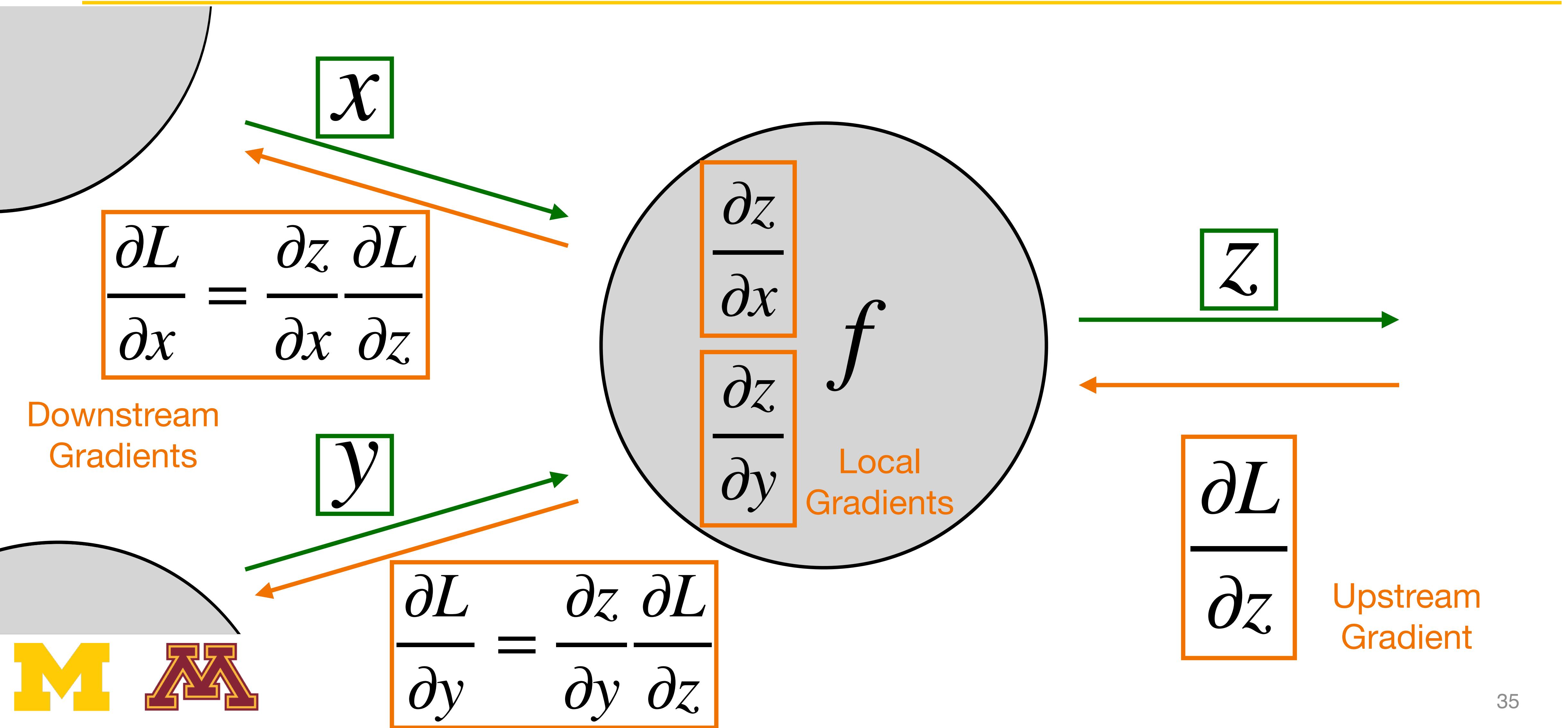
Generalizing Backpropagation



Generalizing Backpropagation



Generalizing Backpropagation

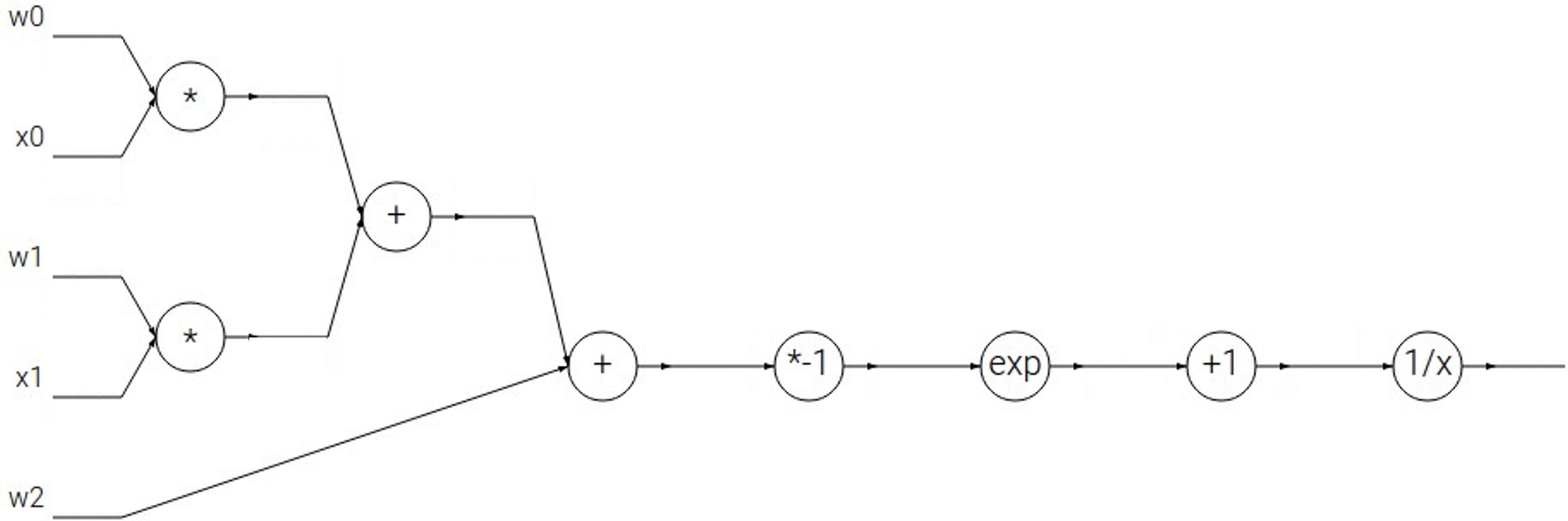


Another example

$$f(x, w) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$$

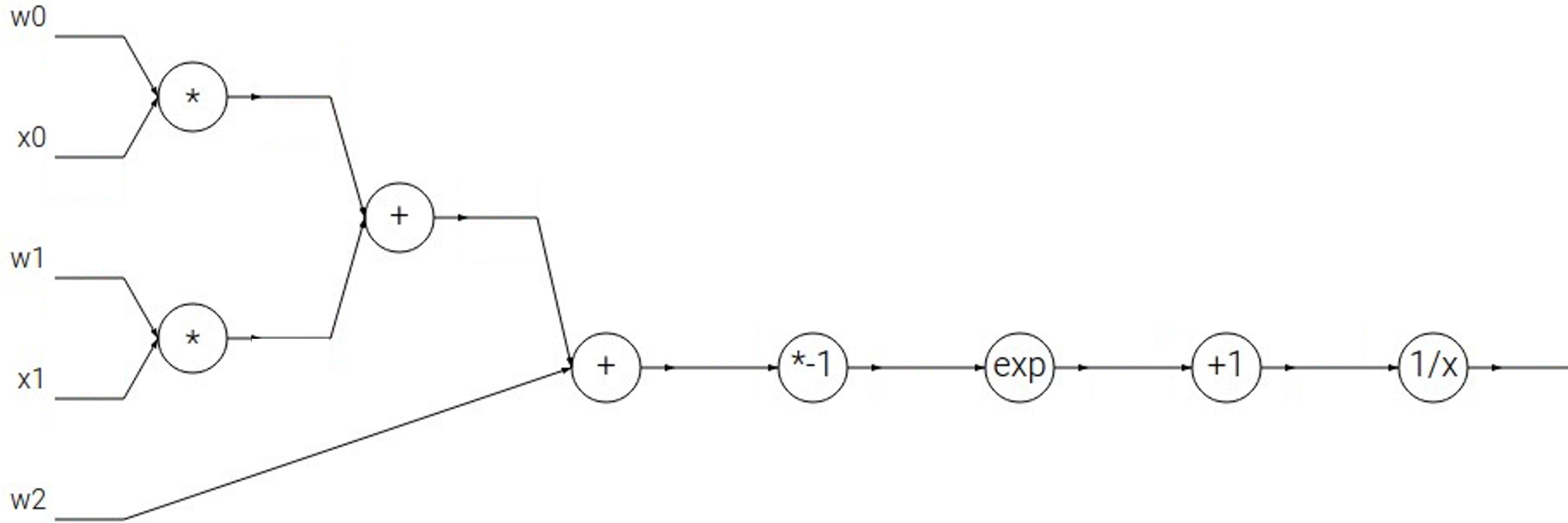
Another example

$$f(x, w) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$$



Another example

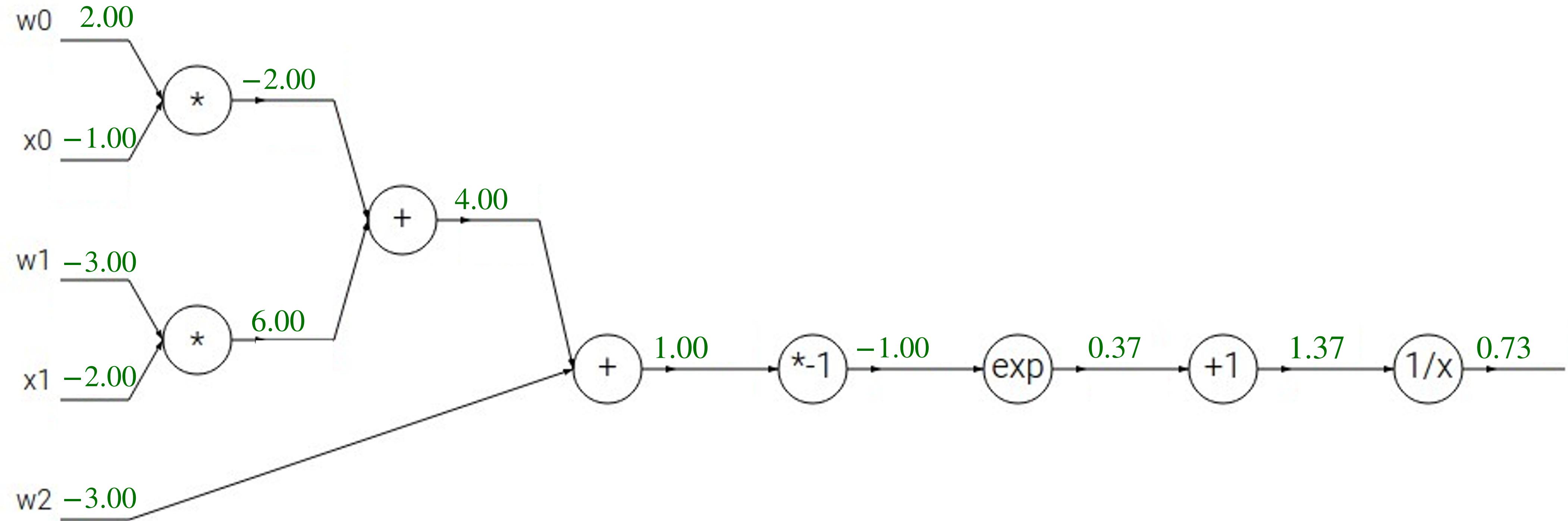
$$f(x, w) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$$



1. Forward pass: Compute outputs

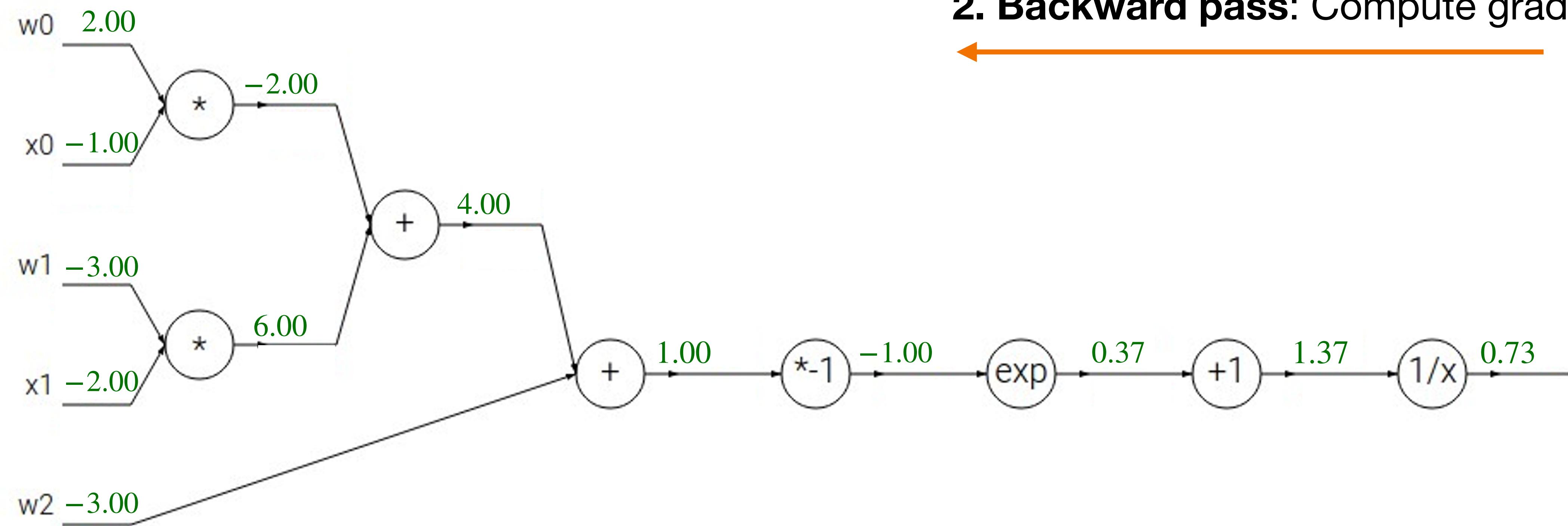
Another example

$$f(x, w) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$$

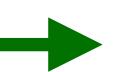


Another example

$$f(x, w) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$$



1. Forward pass: Compute outputs

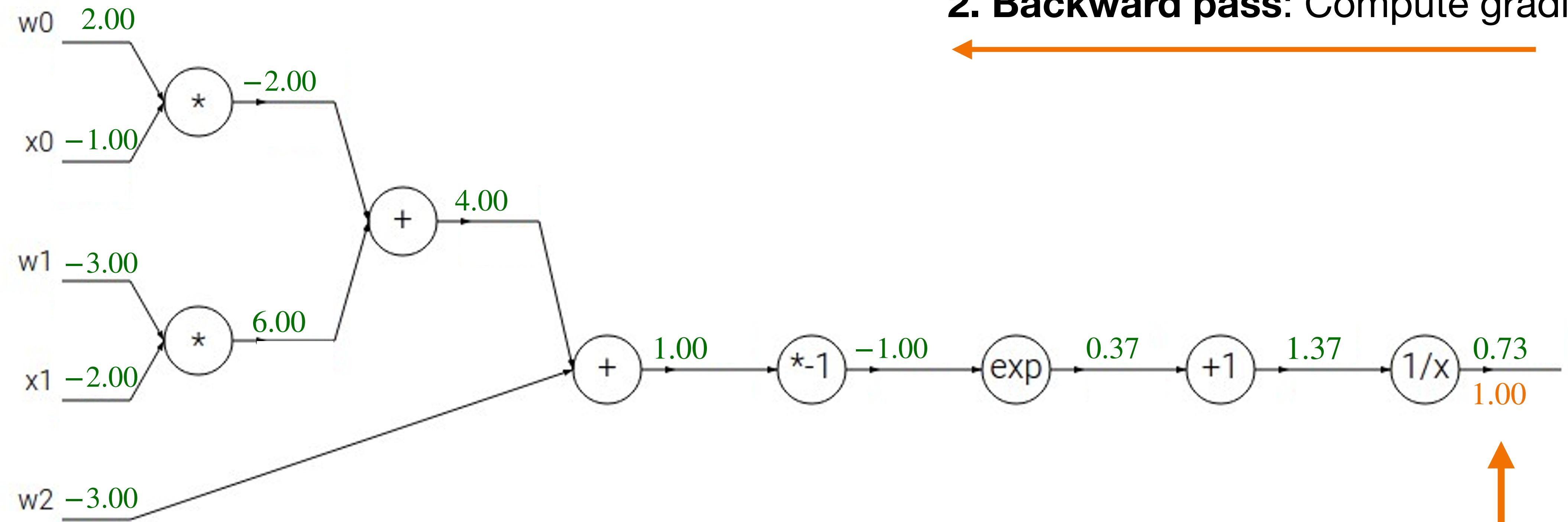


2. Backward pass: Compute gradients



Another example

$$f(x, w) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$$



Another example

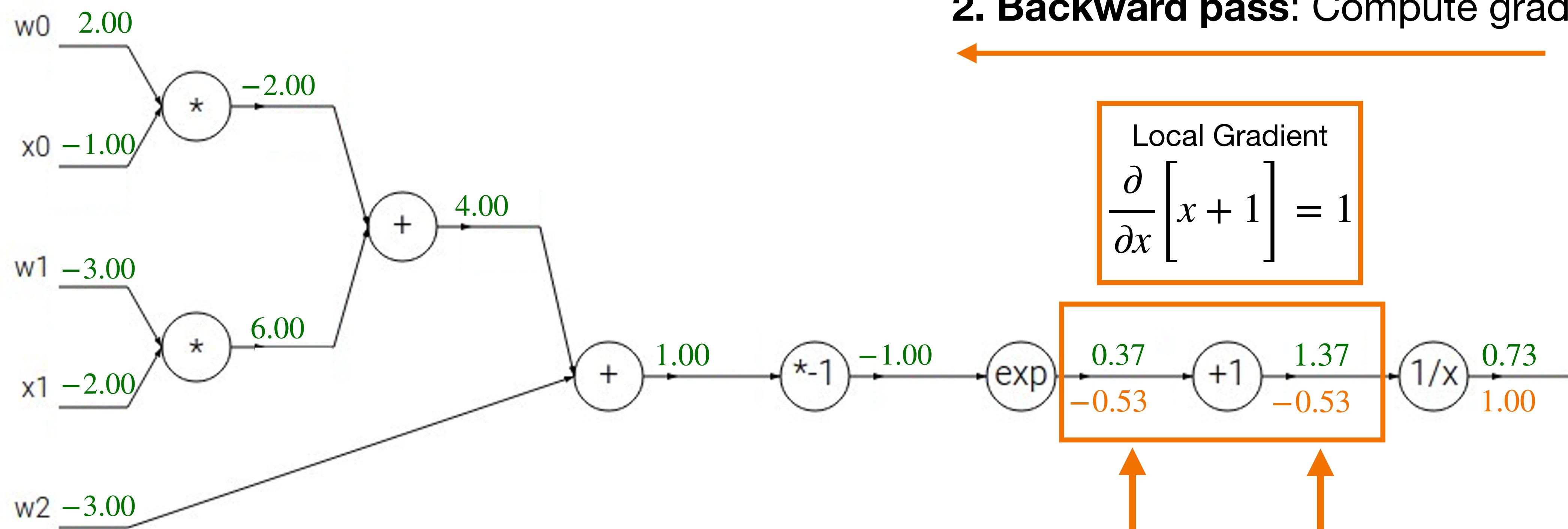
$$f(x, w) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$$

```

graph LR
    x0[2.00] --> M1(( ))
    x1[-1.00] --> M1
    M1 -- "*" --> P1(( ))
    w0[2.00] --> P1
    P1 -- "-" --> S1(( ))
    S1 -- "+" --> P2(( ))
    w1[-3.00] --> P2
    P2 -- "*" --> M2(( ))
    x1[-2.00] --> M2
    M2 -- "6.00" --> P3(( ))
    P3 -- "+" --> S2(( ))
    w2[-3.00] --> S2
    S2 -- "+" --> P4(( ))
    P4 -- "1.00" --> M3(( ))
    M3 -- "-" --> P5(( ))
    P5 -- "1.00" --> M4(( ))
    M4 -- "0.37" --> P6(( ))
    P6 -- "exp" --> M5(( ))
    M5 -- "0.37" --> P7(( ))
    P7 -- "-1.00" --> M6(( ))
    M6 -- "*" --> P8(( ))
    P8 -- "1.00" --> M7(( ))
    M7 -- "+" --> P9(( ))
    P9 -- "0.37" --> M8(( ))
    M8 -- "+" --> P10(( ))
    P10 -- "0.73" --> M9(( ))
    M9 -- "1.00" --> P11(( ))
    P11 -- "1/x" --> M10(( ))
    M10 -- "0.73" --> P12(( ))
    P12 -- "-" --> M11(( ))
    M11 -- "0.53" --> P13(( ))
    P13 -- "1.37" --> M12(( ))
    M12 -- "-" --> P14(( ))
    P14 -- "1.37" --> M13(( ))
    M13 -- "+" --> P15(( ))
    P15 -- "0.73" --> M14(( ))
    M14 -- "-" --> P16(( ))
    P16 -- "0.53" --> M15(( ))
    M15 -- "+" --> P17(( ))
    P17 -- "1.37" --> M16(( ))
    M16 -- "-" --> P18(( ))
    P18 -- "1.37" --> M17(( ))
    M17 -- "+" --> P19(( ))
    P19 -- "0.73" --> M18(( ))
    M18 -- "-" --> P20(( ))
    P20 -- "0.53" --> M19(( ))
    M19 -- "+" --> P21(( ))
    P21 -- "1.37" --> M20(( ))
    M20 -- "-" --> P22(( ))
    P22 -- "1.37" --> M21(( ))
    M21 -- "+" --> P23(( ))
    P23 -- "0.73" --> M22(( ))
    M22 -- "-" --> P24(( ))
    P24 -- "0.53" --> M23(( ))
    M23 -- "+" --> P25(( ))
    P25 -- "1.37" --> M24(( ))
    M24 -- "-" --> P26(( ))
    P26 -- "1.37" --> M25(( ))
    M25 -- "+" --> P27(( ))
    P27 -- "0.73" --> M26(( ))
    M26 -- "-" --> P28(( ))
    P28 -- "0.53" --> M27(( ))
    M27 -- "+" --> P29(( ))
    P29 -- "1.37" --> M28(( ))
    M28 -- "-" --> P30(( ))
    P30 -- "1.37" --> M29(( ))
    M29 -- "+" --> P31(( ))
    P31 -- "0.73" --> M30(( ))
    M30 -- "-" --> P32(( ))
    P32 -- "0.53" --> M31(( ))
    M31 -- "+" --> P33(( ))
    P33 -- "1.37" --> M32(( ))
    M32 -- "-" --> P34(( ))
    P34 -- "1.37" --> M33(( ))
    M33 -- "+" --> P35(( ))
    P35 -- "0.73" --> M34(( ))
    M34 -- "-" --> P36(( ))
    P36 -- "0.53" --> M35(( ))
    M35 -- "+" --> P37(( ))
    P37 -- "1.37" --> M36(( ))
    M36 -- "-" --> P38(( ))
    P38 -- "1.37" --> M37(( ))
    M37 -- "+" --> P39(( ))
    P39 -- "0.73" --> M40(( ))
    M40 -- "-" --> P41(( ))
    P41 -- "0.53" --> M42(( ))
    M42 -- "+" --> P43(( ))
    P43 -- "1.37" --> M44(( ))
    M44 -- "-" --> P45(( ))
    P45 -- "1.37" --> M46(( ))
    M46 -- "+" --> P47(( ))
    P47 -- "0.73" --> M48(( ))
    M48 -- "-" --> P49(( ))
    P49 -- "0.53" --> M50(( ))
    M50 -- "+" --> P51(( ))
    P51 -- "1.37" --> M52(( ))
    M52 -- "-" --> P53(( ))
    P53 -- "1.37" --> M54(( ))
    M54 -- "+" --> P55(( ))
    P55 -- "0.73" --> M56(( ))
    M56 -- "-" --> P57(( ))
    P57 -- "0.53" --> M58(( ))
    M58 -- "+" --> P59(( ))
    P59 -- "1.37" --> M60(( ))
    M60 -- "-" --> P61(( ))
    P61 -- "1.37" --> M62(( ))
    M62 -- "+" --> P63(( ))
    P63 -- "0.73" --> M64(( ))
    M64 -- "-" --> P65(( ))
    P65 -- "0.53" --> M66(( ))
    M66 -- "+" --> P67(( ))
    P67 -- "1.37" --> M68(( ))
    M68 -- "-" --> P69(( ))
    P69 -- "1.37" --> M70(( ))
    M70 -- "+" --> P71(( ))
    P71 -- "0.73" --> M72(( ))
    M72 -- "-" --> P73(( ))
    P73 -- "0.53" --> M74(( ))
    M74 -- "+" --> P75(( ))
    P75 -- "1.37" --> M76(( ))
    M76 -- "-" --> P77(( ))
    P77 -- "1.37" --> M79(( ))
    M79 -- "+" --> P80(( ))
    P80 -- "0.73" --> M82(( ))
    M82 -- "-" --> P84(( ))
    P84 -- "0.53" --> M86(( ))
    M86 -- "+" --> P88(( ))
    P88 -- "1.37" --> M90(( ))
    M90 -- "-" --> P92(( ))
    P92 -- "1.37" --> M94(( ))
    M94 -- "+" --> P96(( ))
    P96 -- "0.73" --> M98(( ))
    M98 -- "-" --> P100(( ))
    P100 -- "0.53" --> M102(( ))
    M102 -- "+" --> P104(( ))
    P104 -- "1.37" --> M106(( ))
    M106 -- "-" --> P108(( ))
    P108 -- "1.37" --> M110(( ))
    M110 -- "+" --> P112(( ))
    P112 -- "0.73" --> M114(( ))
    M114 -- "-" --> P116(( ))
    P116 -- "0.53" --> M118(( ))
    M118 -- "+" --> P120(( ))
    P120 -- "1.37" --> M122(( ))
    M122 -- "-" --> P124(( ))
    P124 -- "1.37" --> M126(( ))
    M126 -- "+" --> P128(( ))
    P128 -- "0.73" --> M130(( ))
    M130 -- "-" --> P132(( ))
    P132 -- "0.53" --> M134(( ))
    M134 -- "+" --> P136(( ))
    P136 -- "1.37" --> M138(( ))
    M138 -- "-" --> P140(( ))
    P140 -- "1.37" --> M142(( ))
    M142 -- "+" --> P144(( ))
    P144 -- "0.73" --> M146(( ))
    M146 -- "-" --> P148(( ))
    P148 -- "0.53" --> M150(( ))
    M150 -- "+" --> P152(( ))
    P152 -- "1.37" --> M154(( ))
    M154 -- "-" --> P156(( ))
    P156 -- "1.37" --> M158(( ))
    M158 -- "+" --> P160(( ))
    P160 -- "0.73" --> M162(( ))
    M162 -- "-" --> P164(( ))
    P164 -- "0.53" --> M166(( ))
    M166 -- "+" --> P168(( ))
    P168 -- "1.37" --> M170(( ))
    M170 -- "-" --> P172(( ))
    P172 -- "1.37" --> M174(( ))
    M174 -- "+" --> P176(( ))
    P176 -- "0.73" --> M178(( ))
    M178 -- "-" --> P180(( ))
    P180 -- "0.53" --> M182(( ))
    M182 -- "+" --> P184(( ))
    P184 -- "1.37" --> M186(( ))
    M186 -- "-" --> P188(( ))
    P188 -- "1.37" --> M190(( ))
    M190 -- "+" --> P192(( ))
    P192 -- "0.73" --> M194(( ))
    M194 -- "-" --> P196(( ))
    P196 -- "0.53" --> M198(( ))
    M198 -- "+" --> P200(( ))
    P200 -- "1.37" --> M202(( ))
    M202 -- "-" --> P204(( ))
    P204 -- "1.37" --> M206(( ))
    M206 -- "+" --> P208(( ))
    P208 -- "0.73" --> M210(( ))
    M210 -- "-" --> P212(( ))
    P212 -- "0.53" --> M214(( ))
    M214 -- "+" --> P216(( ))
    P216 -- "1.37" --> M218(( ))
    M218 -- "-" --> P220(( ))
    P220 -- "1.37" --> M222(( ))
    M222 -- "+" --> P224(( ))
    P224 -- "0.73" --> M226(( ))
    M226 -- "-" --> P230(( ))
    P230 -- "0.53" --> M232(( ))
    M232 -- "+" --> P234(( ))
    P234 -- "1.37" --> M236(( ))
    M236 -- "-" --> P238(( ))
    P238 -- "1.37" --> M240(( ))
    M240 -- "+" --> P242(( ))
    P242 -- "0.73" --> M244(( ))
    M244 -- "-" --> P248(( ))
    P248 -- "0.53" --> M250(( ))
    M250 -- "+" --> P252(( ))
    P252 -- "1.37" --> M254(( ))
    M254 -- "-" --> P256(( ))
    P256 -- "1.37" --> M258(( ))
    M258 -- "+" --> P260(( ))
    P260 -- "0.73" --> M262(( ))
    M262 -- "-" --> P264(( ))
    P264 -- "0.53" --> M266(( ))
    M266 -- "+" --> P268(( ))
    P268 -- "1.37" --> M270(( ))
    M270 -- "-" --> P272(( ))
    P272 -- "1.37" --> M274(( ))
    M274 -- "+" --> P276(( ))
    P276 -- "0.73" --> M278(( ))
    M278 -- "-" --> P280(( ))
    P280 -- "0.53" --> M282(( ))
    M282 -- "+" --> P284(( ))
    P284 -- "1.37" --> M286(( ))
    M286 -- "-" --> P290(( ))
    P290 -- "1.37" --> M292(( ))
    M292 -- "+" --> P294(( ))
    P294 -- "0.73" --> M296(( ))
    M296 -- "-" --> P298(( ))
    P298 -- "0.53" --> M300(( ))
    M300 -- "+" --> P302(( ))
    P302 -- "1.37" --> M304(( ))
    M304 -- "-" --> P308(( ))
    P308 -- "1.37" --> M310(( ))
    M310 -- "+" --> P312(( ))
    P312 -- "0.73" --> M314(( ))
    M314 -- "-" --> P316(( ))
    P316 -- "0.53" --> M318(( ))
    M318 -- "+" --> P320(( ))
    P320 -- "1.37" --> M322(( ))
    M322 -- "-" --> P324(( ))
    P324 -- "1.37" --> M326(( ))
    M326 -- "+" --> P328(( ))
    P328 -- "0.73" --> M330(( ))
    M330 -- "-" --> P332(( ))
    P332 -- "0.53" --> M334(( ))
    M334 -- "+" --> P336(( ))
    P336 -- "1.37" --> M338(( ))
    M338 -- "-" --> P340(( ))
    P340 -- "1.37" --> M342(( ))
    M342 -- "+" --> P344(( ))
    P344 -- "0.73" --> M346(( ))
    M346 -- "-" --> P348(( ))
    P348 -- "0.53" --> M350(( ))
    M350 -- "+" --> P352(( ))
    P352 -- "1.37" --> M354(( ))
    M354 -- "-" --> P356(( ))
    P356 -- "1.37" --> M358(( ))
    M358 -- "+" --> P360(( ))
    P360 -- "0.73" --> M362(( ))
    M362 -- "-" --> P364(( ))
    P364 -- "0.53" --> M366(( ))
    M366 -- "+" --> P368(( ))
    P368 -- "1.37" --> M370(( ))
    M370 -- "-" --> P372(( ))
    P372 -- "1.37" --> M374(( ))
    M374 -- "+" --> P376(( ))
    P376 -- "0.73" --> M378(( ))
    M378 -- "-" --> P380(( ))
    P380 -- "0.53" --> M382(( ))
    M382 -- "+" --> P384(( ))
    P384 -- "1.37" --> M386(( ))
    M386 -- "-" --> P390(( ))
    P390 -- "1.37" --> M392(( ))
    M392 -- "+" --> P394(( ))
    P394 -- "0.73" --> M396(( ))
    M396 -- "-" --> P398(( ))
    P398 -- "0.53" --> M400(( ))
    M400 -- "+" --> P402(( ))
    P402 -- "1.37" --> M404(( ))
    M404 -- "-" --> P408(( ))
    P408 -- "1.37" --> M410(( ))
    M410 -- "+" --> P412(( ))
    P412 -- "0.73" --> M414(( ))
    M414 -- "-" --> P416(( ))
    P416 -- "0.53" --> M418(( ))
    M418 -- "+" --> P420(( ))
    P420 -- "1.37" --> M422(( ))
    M422 -- "-" --> P424(( ))
    P424 -- "1.37" --> M426(( ))
    M426 -- "+" --> P428(( ))
    P428 -- "0.73" --> M430(( ))
    M430 -- "-" --> P432(( ))
    P432 -- "0.53" --> M434(( ))
    M434 -- "+" --> P436(( ))
    P436 -- "1.37" --> M438(( ))
    M438 -- "-" --> P440(( ))
    P440 -- "1.37" --> M442(( ))
    M442 -- "+" --> P444(( ))
    P444 -- "0.73" --> M446(( ))
    M446 -- "-" --> P448(( ))
    P448 -- "0.53" --> M450(( ))
    M450 -- "+" --> P452(( ))
    P452 -- "1.37" --> M454(( ))
    M454 -- "-" --> P456(( ))
    P456 -- "1.37" --> M458(( ))
    M458 -- "+" --> P460(( ))
    P460 -- "0.73" --> M462(( ))
    M462 -- "-" --> P464(( ))
    P464 -- "0.53" --> M466(( ))
    M466 -- "+" --> P468(( ))
    P468 -- "1.37" --> M470(( ))
    M470 -- "-" --> P472(( ))
    P472 -- "1.37" --> M474(( ))
    M474 -- "+" --> P476(( ))
    P476 -- "0.73" --> M478(( ))
    M478 -- "-" --> P480(( ))
    P480 -- "0.53" --> M482(( ))
    M482 -- "+" --> P484(( ))
    P484 -- "1.37" --> M486(( ))
    M486 -- "-" --> P490(( ))
    P490 -- "1.37" --> M492(( ))
    M492 -- "+" --> P494(( ))
    P494 -- "0.73" --> M496(( ))
    M496 -- "-" --> P498(( ))
    P498 -- "0.53" --> M500(( ))
    M500 -- "+" --> P502(( ))
    P502 -- "1.37" --> M504(( ))
    M504 -- "-" --> P508(( ))
    P508 -- "1.37" --> M510(( ))
    M510 -- "+" --> P512(( ))
    P512 -- "0.73" --> M514(( ))
    M514 -- "-" --> P516(( ))
    P516 -- "0.53" --> M518(( ))
    M518 -- "+" --> P520(( ))
    P520 -- "1.37" --> M522(( ))
    M522 -- "-" --> P524(( ))
    P524 -- "1.37" --> M526(( ))
    M526 -- "+" --> P528(( ))
    P528 -- "0.73" --> M530(( ))
    M530 -- "-" --> P532(( ))
    P532 -- "0.53" --> M534(( ))
    M534 -- "+" --> P536(( ))
    P536 -- "1.37" --> M538(( ))
    M538 -- "-" --> P540(( ))
    P540 -- "1.37" --> M542(( ))
    M542 -- "+" --> P544(( ))
    P544 -- "0.73" --> M546(( ))
    M546 -- "-" --> P548(( ))
    P548 -- "0.53" --> M550(( ))
    M550 -- "+" --> P552(( ))
    P552 -- "1.37" --> M554(( ))
    M554 -- "-" --> P556(( ))
    P556 -- "1.37" --> M558(( ))
    M558 -- "+" --> P560(( ))
    P560 -- "0.73" --> M562(( ))
    M562 -- "-" --> P564(( ))
    P564 -- "0.53" --> M566(( ))
    M566 -- "+" --> P568(( ))
    P568 -- "1.37" --> M570(( ))
    M570 -- "-" --> P572(( ))
    P572 -- "1.37" --> M574(( ))
    M574 -- "+" --> P576(( ))
    P576 -- "0.73" --> M578(( ))
    M578 -- "-" --> P580(( ))
    P580 -- "0.53" --> M582(( ))
    M582 -- "+" --> P584(( ))
    P584 -- "1.37" --> M586(( ))
    M586 -- "-" --> P590(( ))
    P590 -- "1.37" --> M592(( ))
    M592 -- "+" --> P594(( ))
    P594 -- "0.73" --> M596(( ))
    M596 -- "-" --> P598(( ))
    P598 -- "0.53" --> M600(( ))
    M600 -- "+" --> P602(( ))
    P602 -- "1.37" --> M604(( ))
    M604 -- "-" --> P608(( ))
    P608 -- "1.37" --> M610(( ))
    M610 -- "+" --> P612(( ))
    P612 -- "0.73" --> M614(( ))
    M614 -- "-" --> P616(( ))
    P616 -- "0.53" --> M618(( ))
    M618 -- "+" --> P620(( ))
    P620 -- "1.37" --> M622(( ))
    M622 -- "-" --> P624(( ))
    P624 -- "1.37" --> M626(( ))
    M626 -- "+" --> P628(( ))
    P628 -- "0.73" --> M630(( ))
    M630 -- "-" --> P632(( ))
    P632 -- "0.53" --> M634(( ))
    M634 -- "+" --> P636(( ))
    P636 -- "1.37" --> M638(( ))
    M638 -- "-" --> P640(( ))
    P640 -- "1.37" --> M642(( ))
    M642 -- "+" --> P644(( ))
    P644 -- "0.73" --> M646(( ))
    M646 -- "-" --> P648(( ))
    P648 -- "0.53" --> M650(( ))
    M650 -- "+" --> P652(( ))
    P652 -- "1.37" --> M654(( ))
    M654 -- "-" --> P656(( ))
    P656 --
```

Another example

$$f(x, w) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$$



1. Forward pass: Compute outputs

2. Backward pass: Compute gradients

$$\frac{\partial}{\partial x} [x + 1] =$$

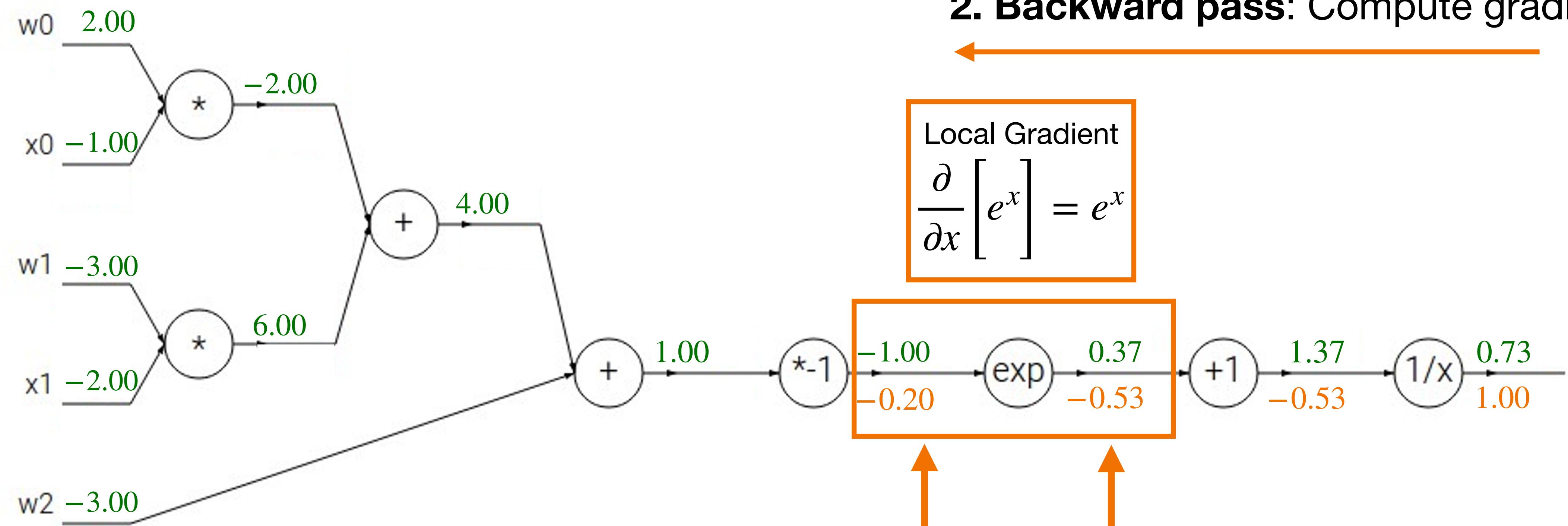
Downstream Gradient

Upstream Gradient



Another example

$$f(x, w) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$$



Another example

$$f(x, w) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$$

$$\begin{matrix} w_0 & 2.00 \\ x_0 & -1.00 \end{matrix}$$

$$\begin{matrix} w_1 & -3.00 \\ x_1 & -2.00 \end{matrix}$$

$$\begin{matrix} w_2 & -3.00 \end{matrix}$$

Local Gradient

$$\frac{\partial}{\partial x} [-1 \cdot x] = -1$$

$$\begin{matrix} 1.00 & -1.00 \\ 0.20 & -0.20 \end{matrix}$$

Downstream
Gradient

1. Forward pass: Compute outputs

2. Backward pass: Compute gradients



$$\begin{matrix} *-1 & \exp \\ -1.00 & -0.53 \end{matrix}$$

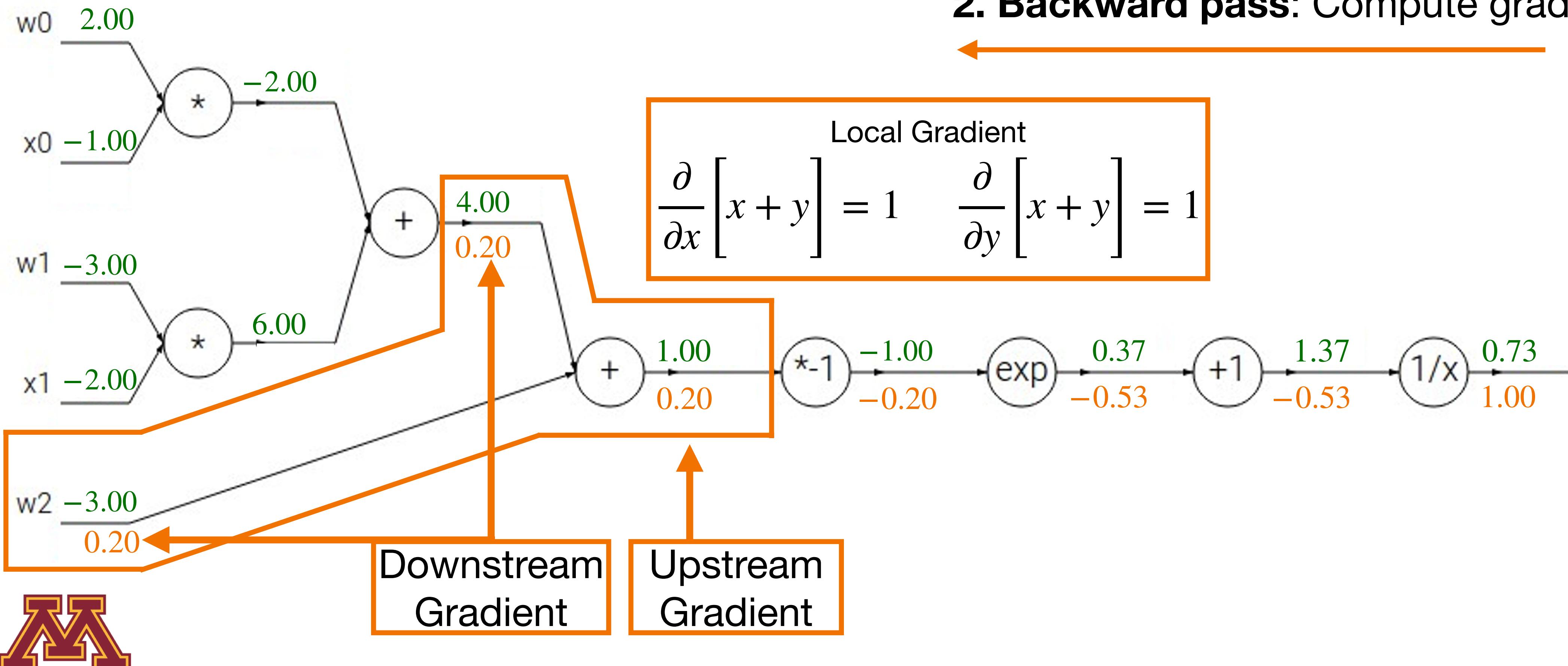
Upstream
Gradient

$$\begin{matrix} 0.37 & 1.37 \\ -0.53 & -0.53 \end{matrix}$$

$$\begin{matrix} 0.73 & 1.00 \end{matrix}$$

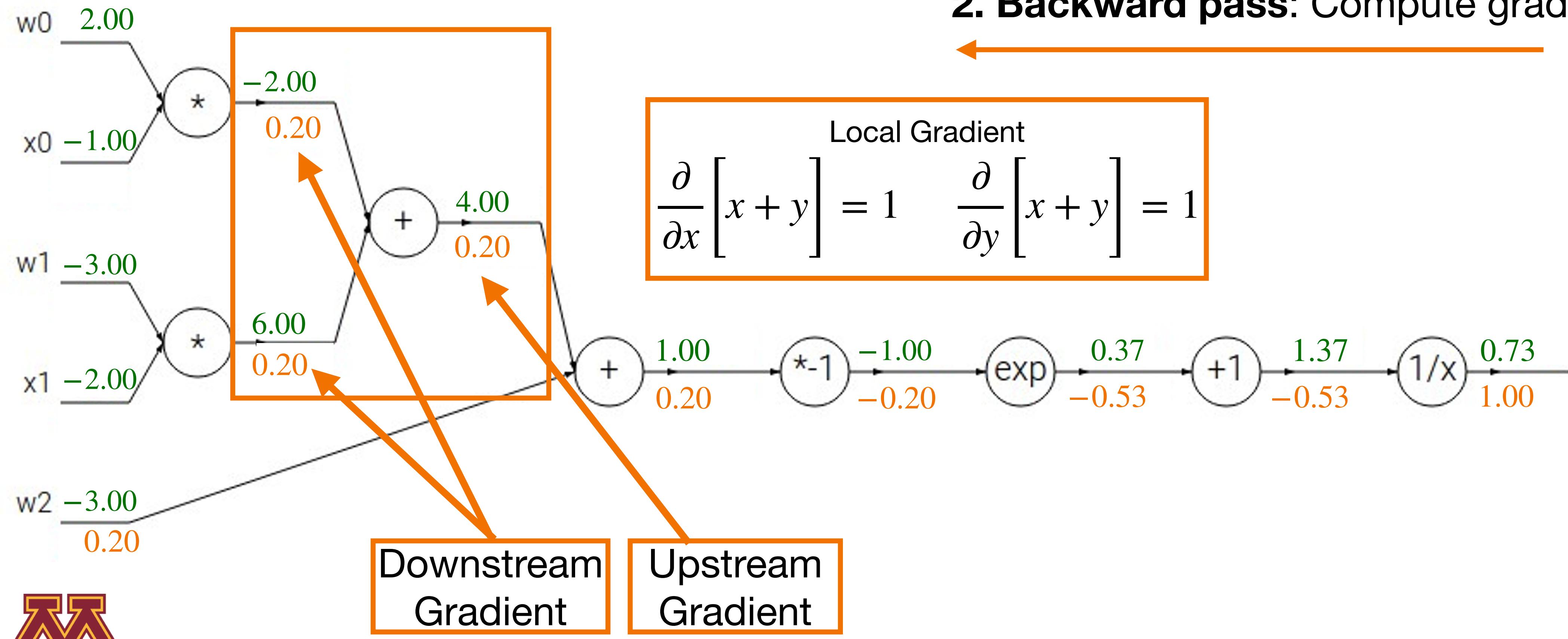
Another example

$$f(x, w) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$$



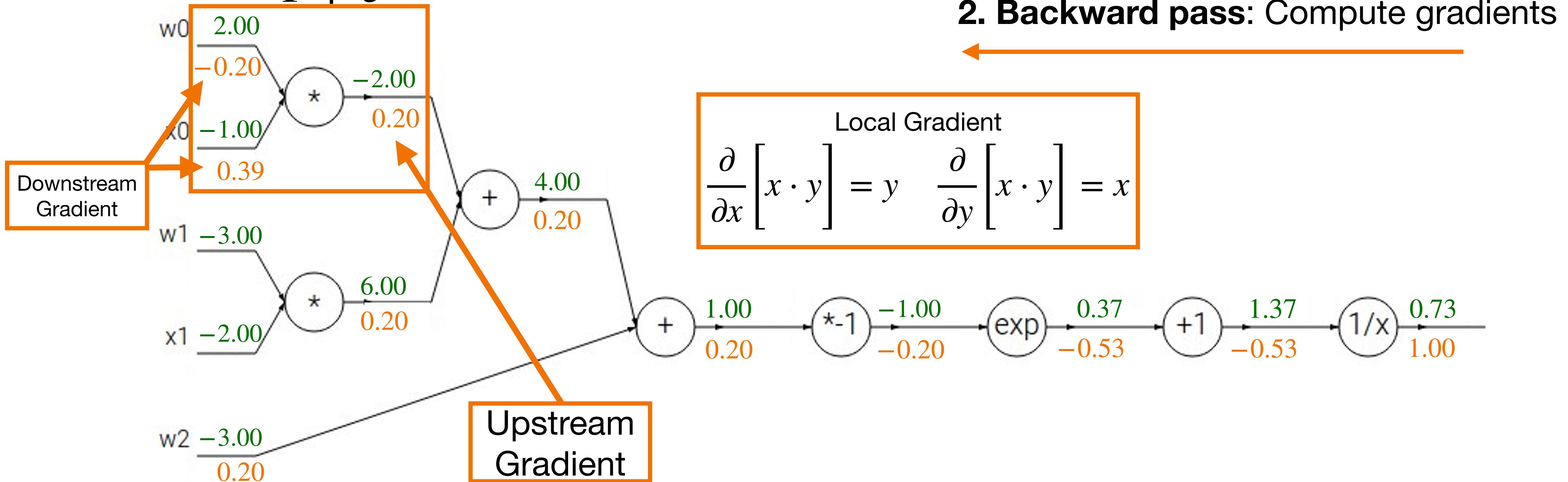
Another example

$$f(x, w) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$$



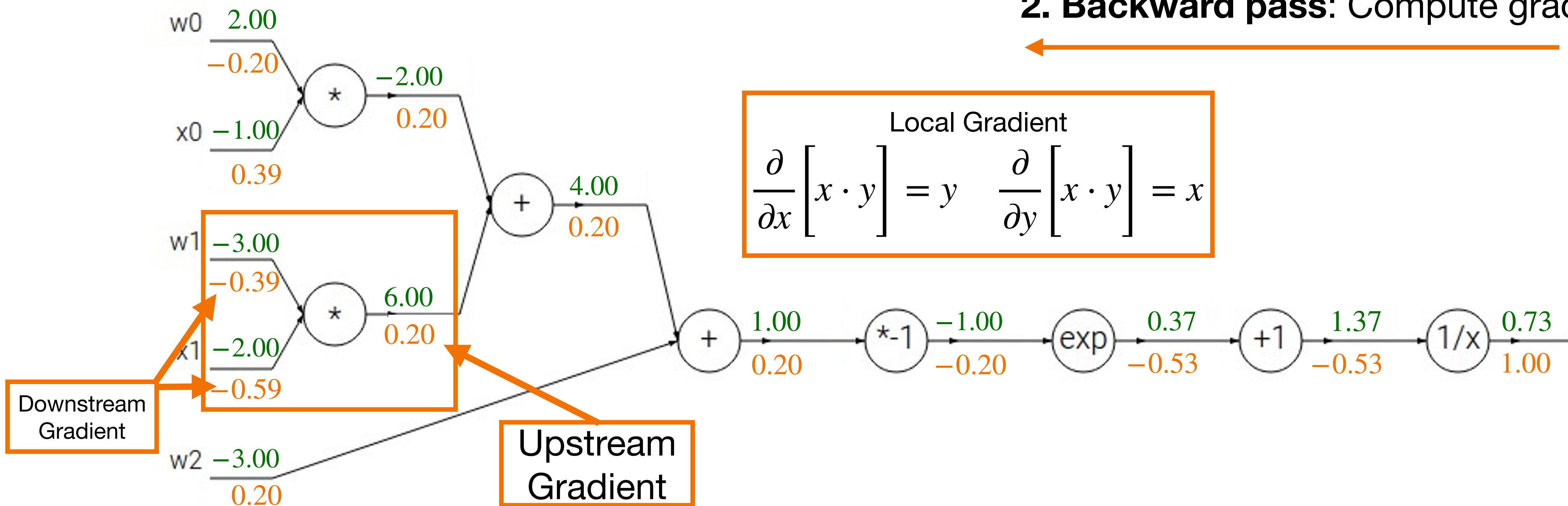
Another example

$$f(x, w) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$$



Another example

$$f(x, w) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$$



1. Forward pass: Compute outputs

2. Backward pass: Compute gradients

Another example

The diagram illustrates the computation of a node's output. The input vector $x = [x_0, x_1]$ is multiplied by the weight vector $w = [w_0, w_1, w_2]$ using element-wise multiplication (Hadamard product). The result is then summed with the bias $b = 1$ to produce the final output $f(x, w)$.

Inputs: $x_0 = -1.00$, $x_1 = -2.00$

Weights: $w_0 = 2.00$, $w_1 = -3.00$, $w_2 = -3.00$

Bias: $b = 1$

Intermediate Calculations:

- Product of x_0 and w_0 : $-1.00 \times 2.00 = -2.00$
- Product of x_1 and w_1 : $-2.00 \times -3.00 = 6.00$
- Product of x_0 and w_2 : $-1.00 \times -3.00 = 3.00$
- Sum of products: $-2.00 + 6.00 + 3.00 = 7.00$
- Addition of bias: $7.00 + 1 = 8.00$

Sigmoid Function: $\sigma(x) = \frac{1}{1 + e^{-x}}$

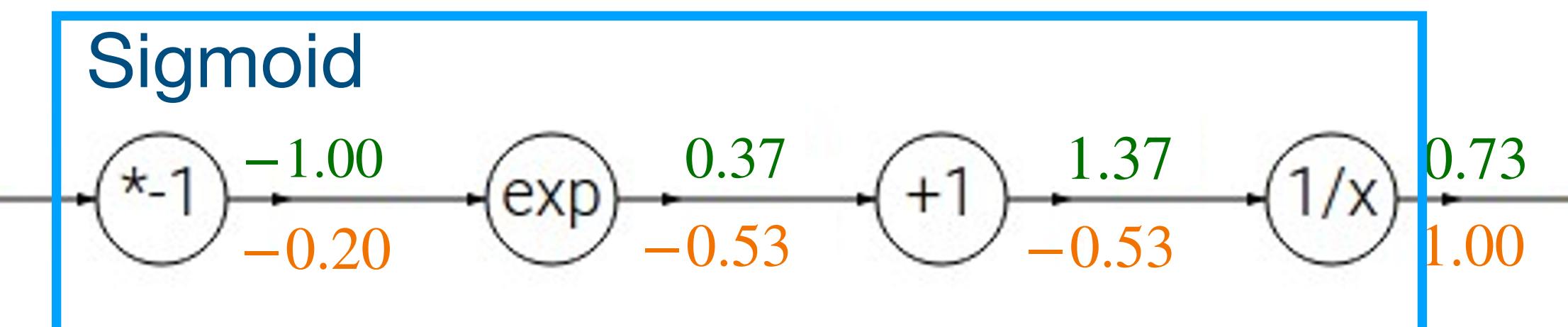
Final Output: $f(x, w) = \sigma(8.00) = 0.999$

1. Forward pass: Compute outputs

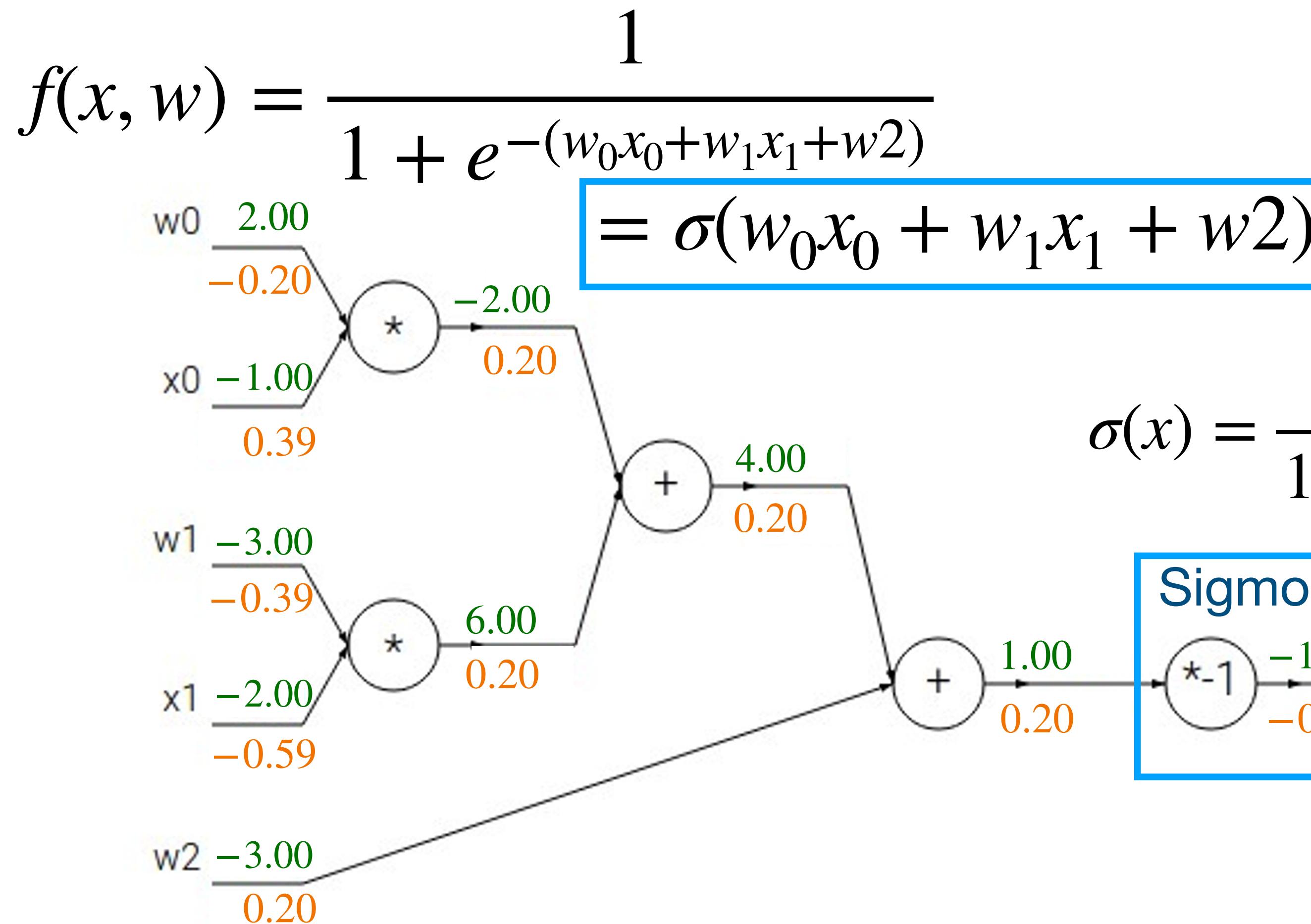
2. Backward pass: Compute gradients

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

Computational graph is not unique: we can use primitives that have simple local gradients



Another example



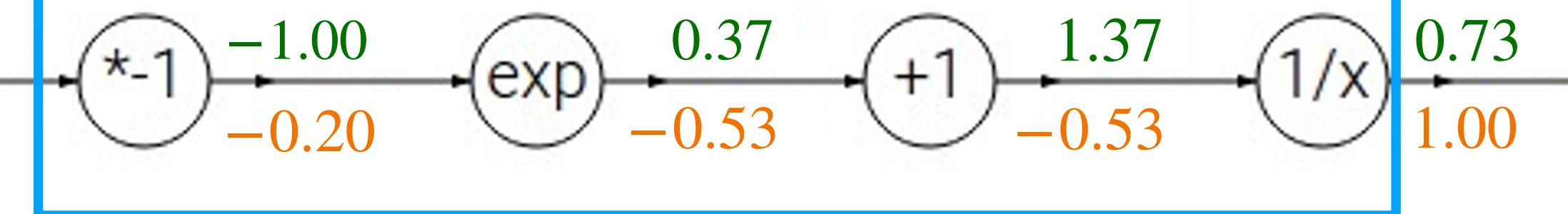
1. Forward pass: Compute outputs

2. Backward pass: Compute gradients

Computational graph is not unique: we can use primitives that have simple local gradients

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

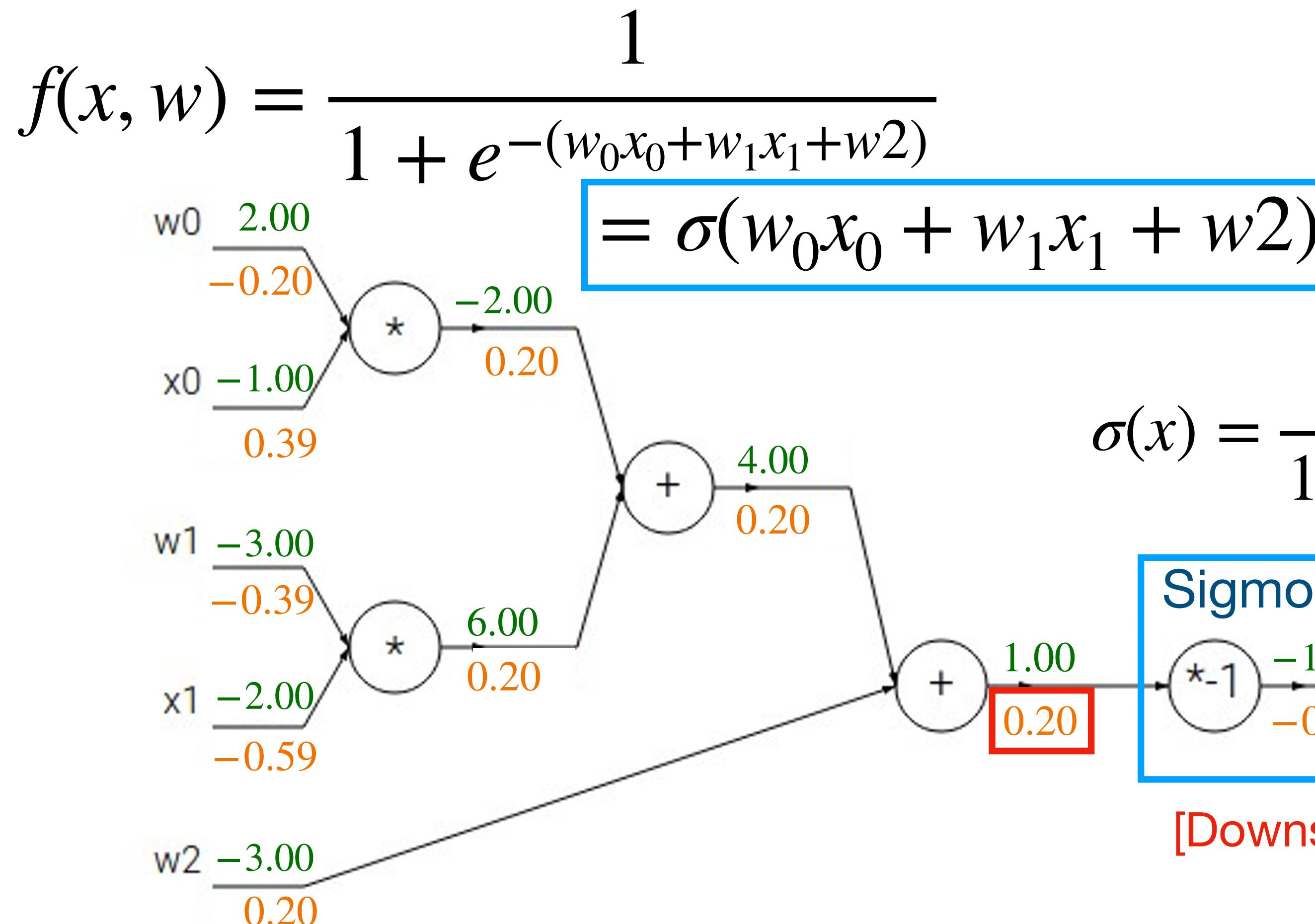
Sigmoid



Sigmoid local gradient:

$$\frac{\partial}{\partial x} [\sigma(x)] = \frac{e^{-x}}{(1 + e^{-x})^2} = \left(\frac{1 + e^{-x} - 1}{1 + e^{-x}} \right) \left(\frac{1}{1 + e^{-x}} \right) = (1 - \sigma(x))\sigma(x)$$

Another example



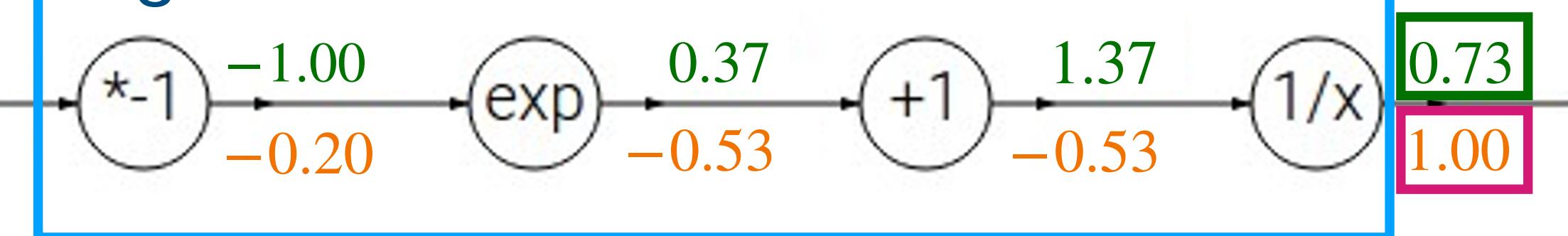
1. Forward pass: Compute outputs

2. Backward pass: Compute gradients

Computational graph is not unique: we can use primitives that have simple local gradients

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

Sigmoid



[Downstream] = [Local] · [Upstream]

$$= (1 - 0.73) \cdot 0.73 \cdot 1.00 = 0.20$$

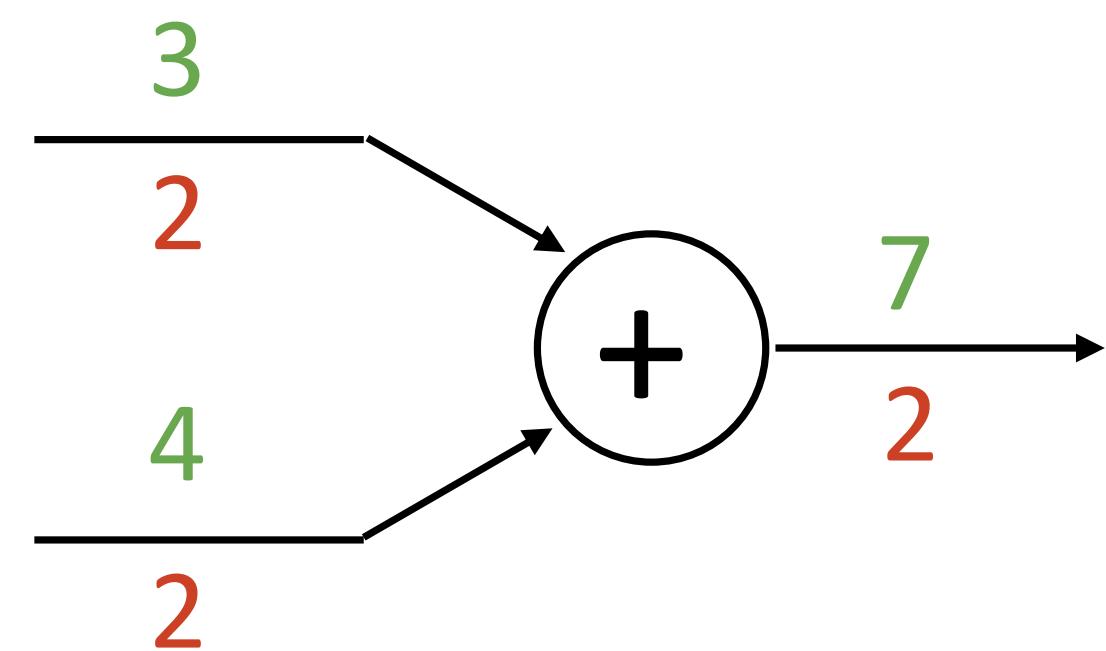


Sigmoid local
gradient:

$$\frac{\partial}{\partial x} [\sigma(x)] = \frac{e^{-x}}{(1 + e^{-x})^2} = \left(\frac{1 + e^{-x} - 1}{1 + e^{-x}} \right) \left(\frac{1}{1 + e^{-x}} \right) = (1 - \sigma(x))\sigma(x)$$

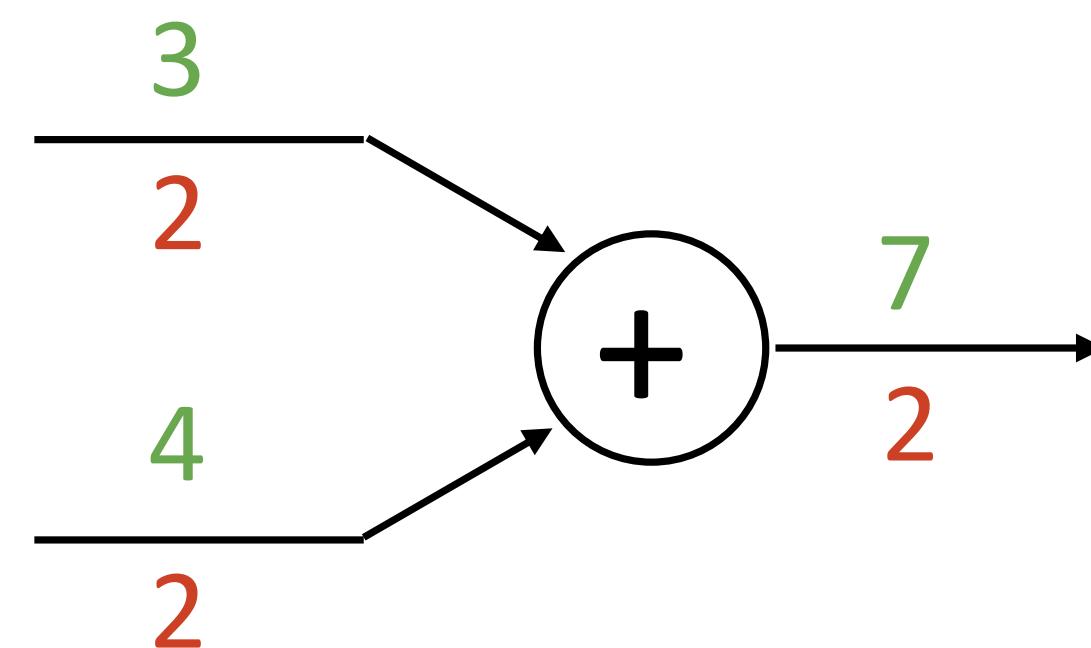
Patterns in Gradient Flow

add gate: gradient distributor

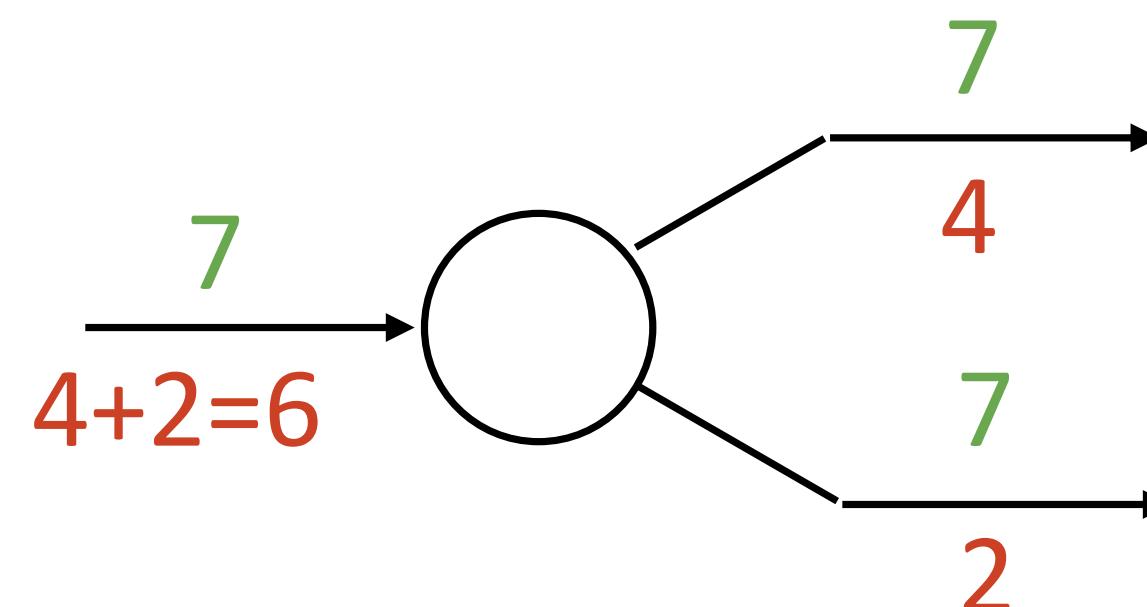


Patterns in Gradient Flow

add gate: gradient distributor

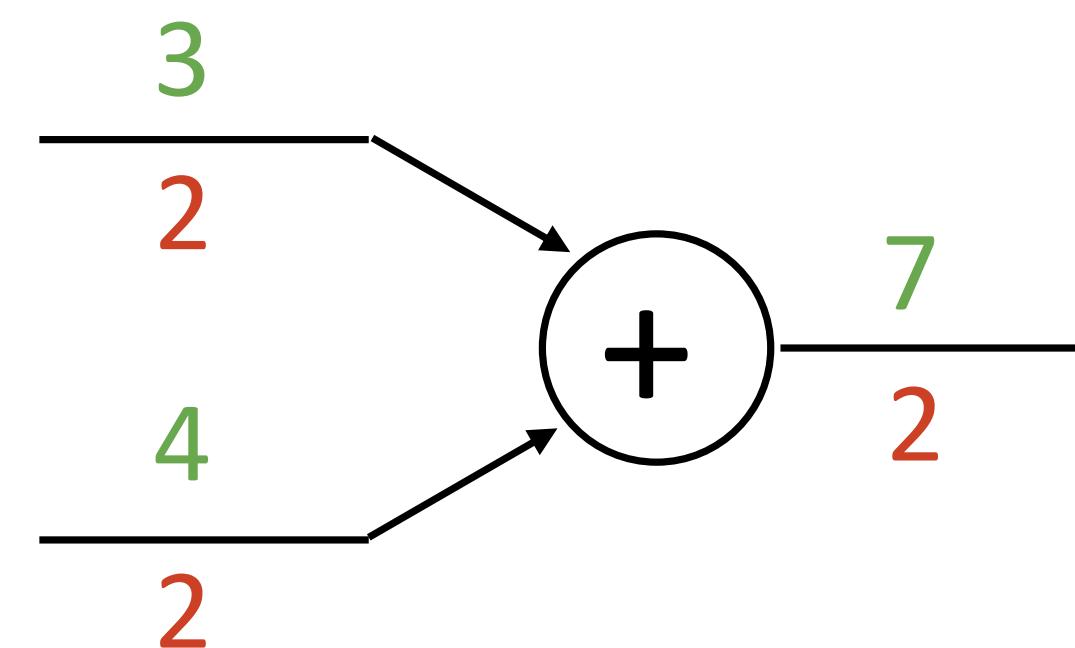


copy gate: gradient adder

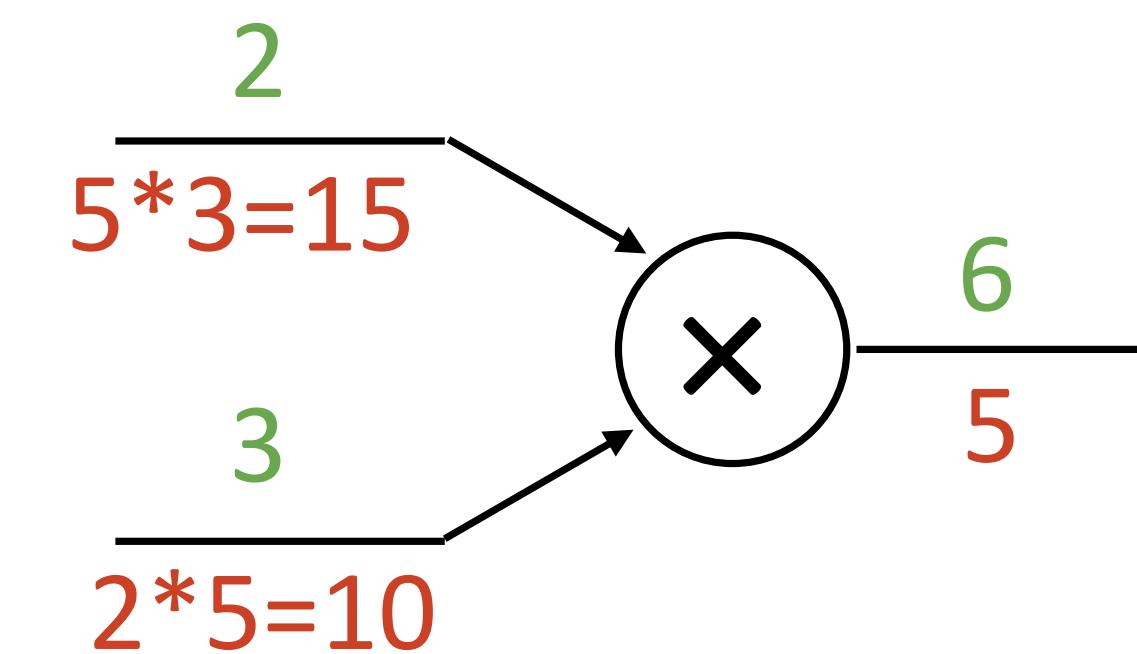


Patterns in Gradient Flow

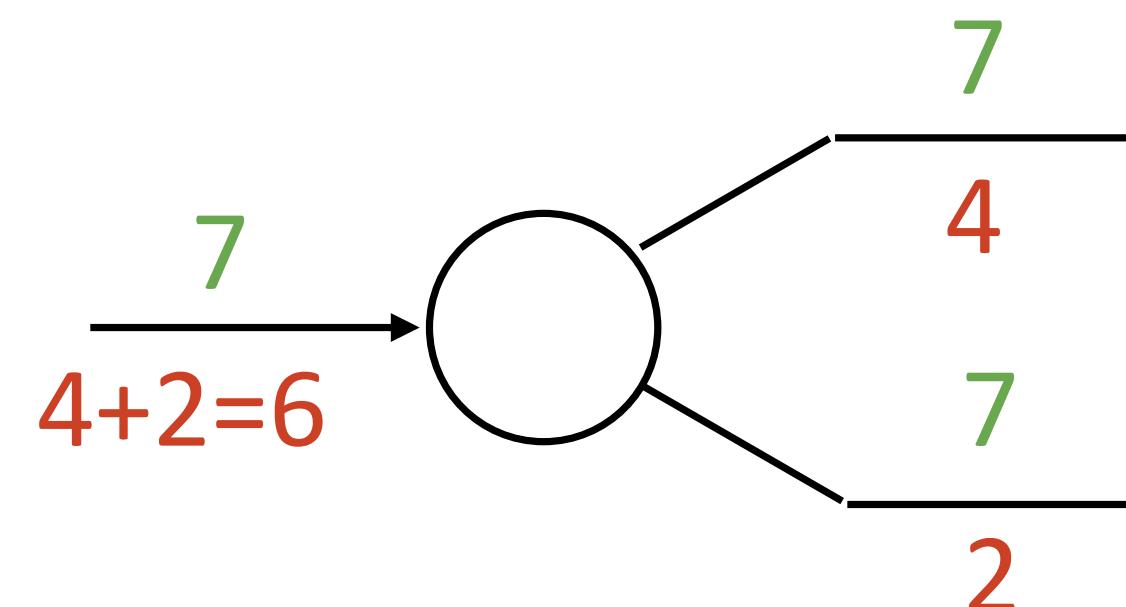
add gate: gradient distributor



mul gate: “swap multiplier”

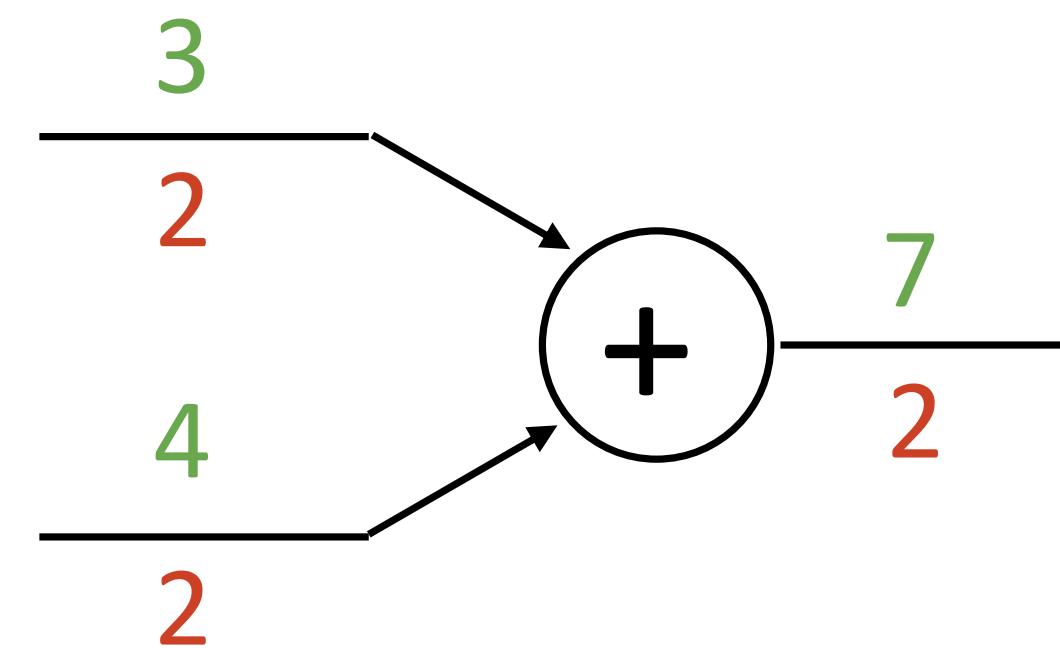


copy gate: gradient adder

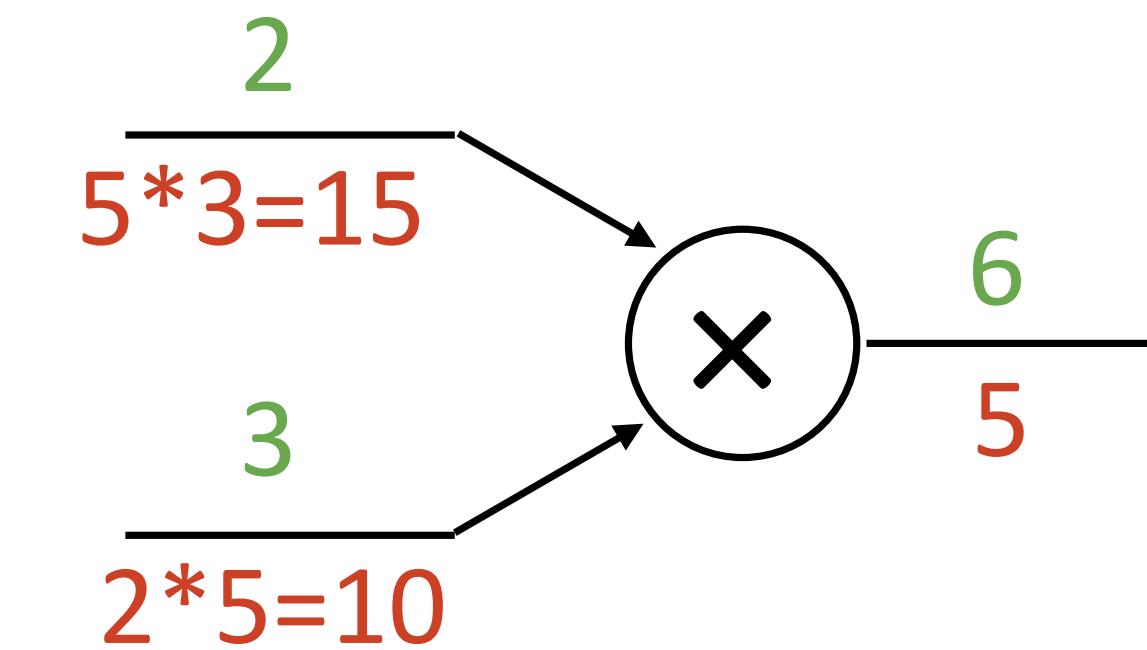


Patterns in Gradient Flow

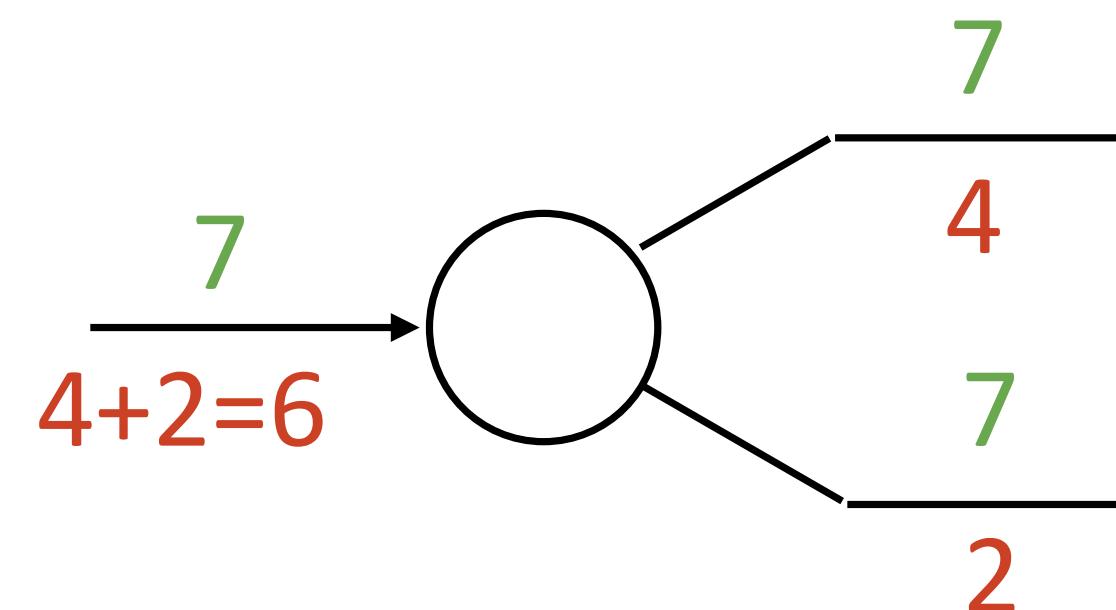
add gate: gradient distributor



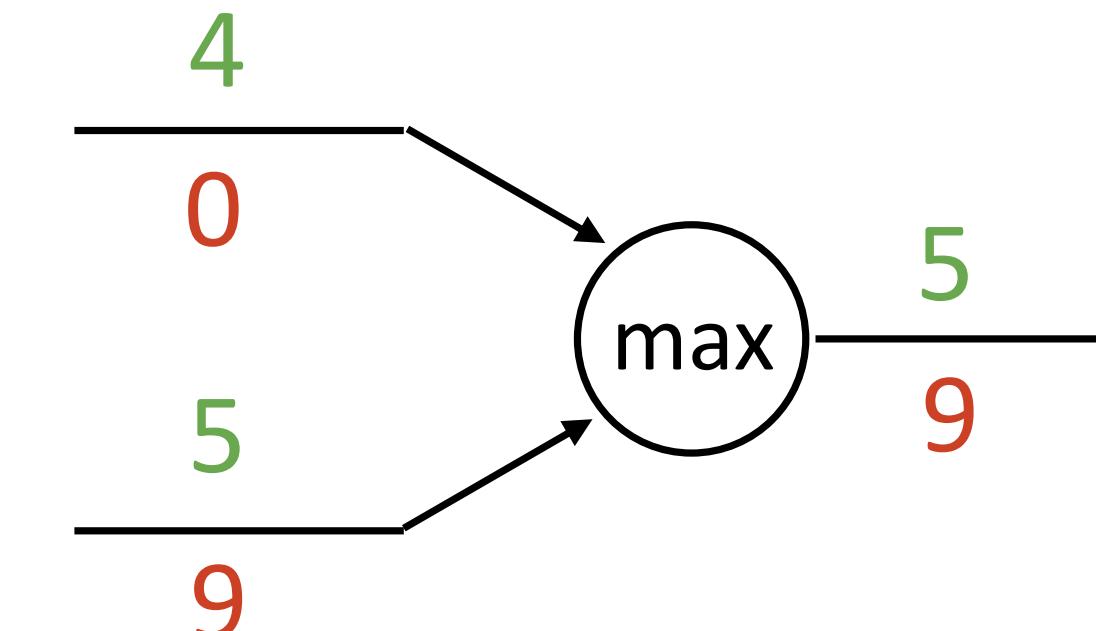
mul gate: “swap multiplier”



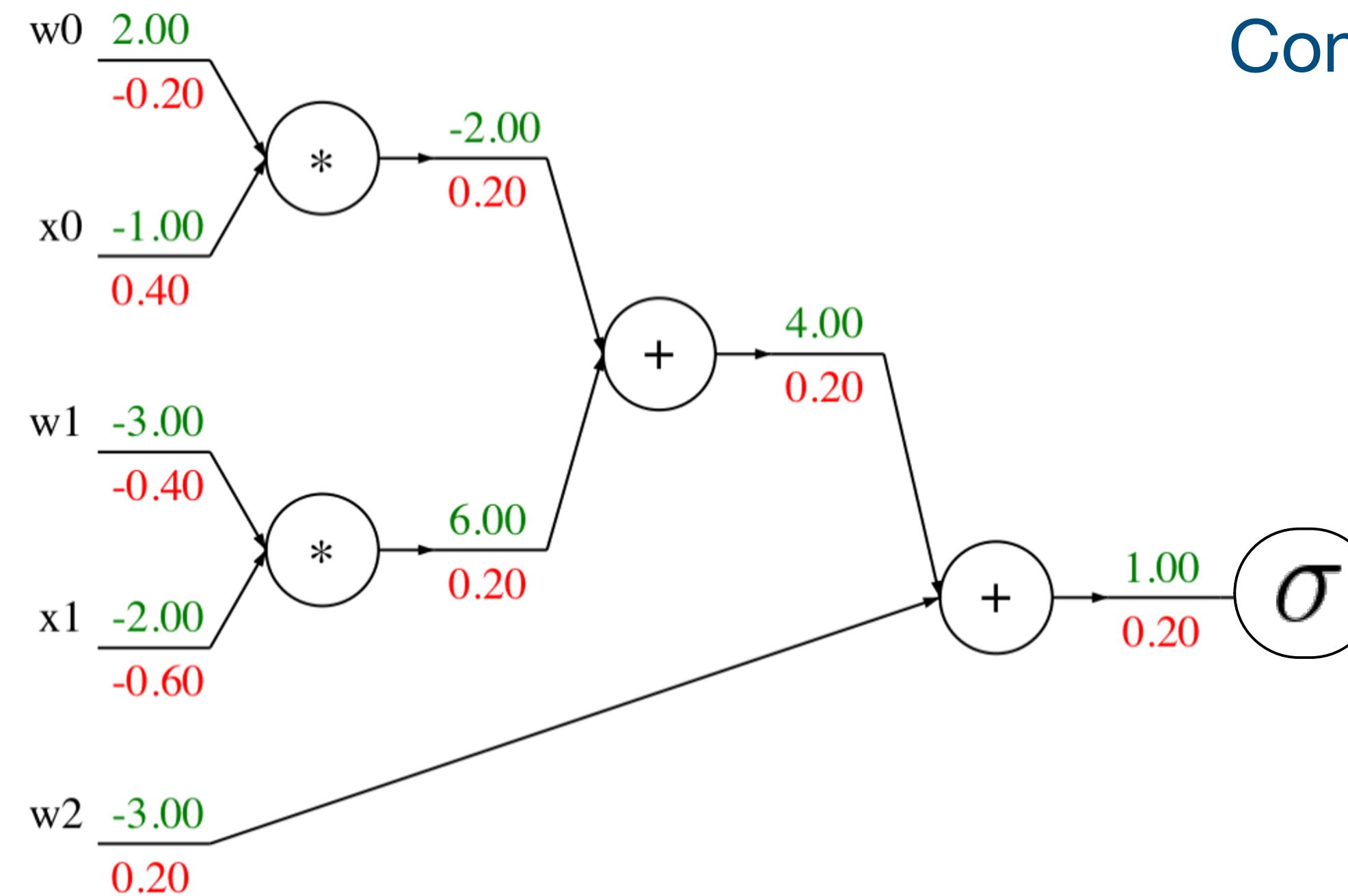
copy gate: gradient adder



max gate: gradient router



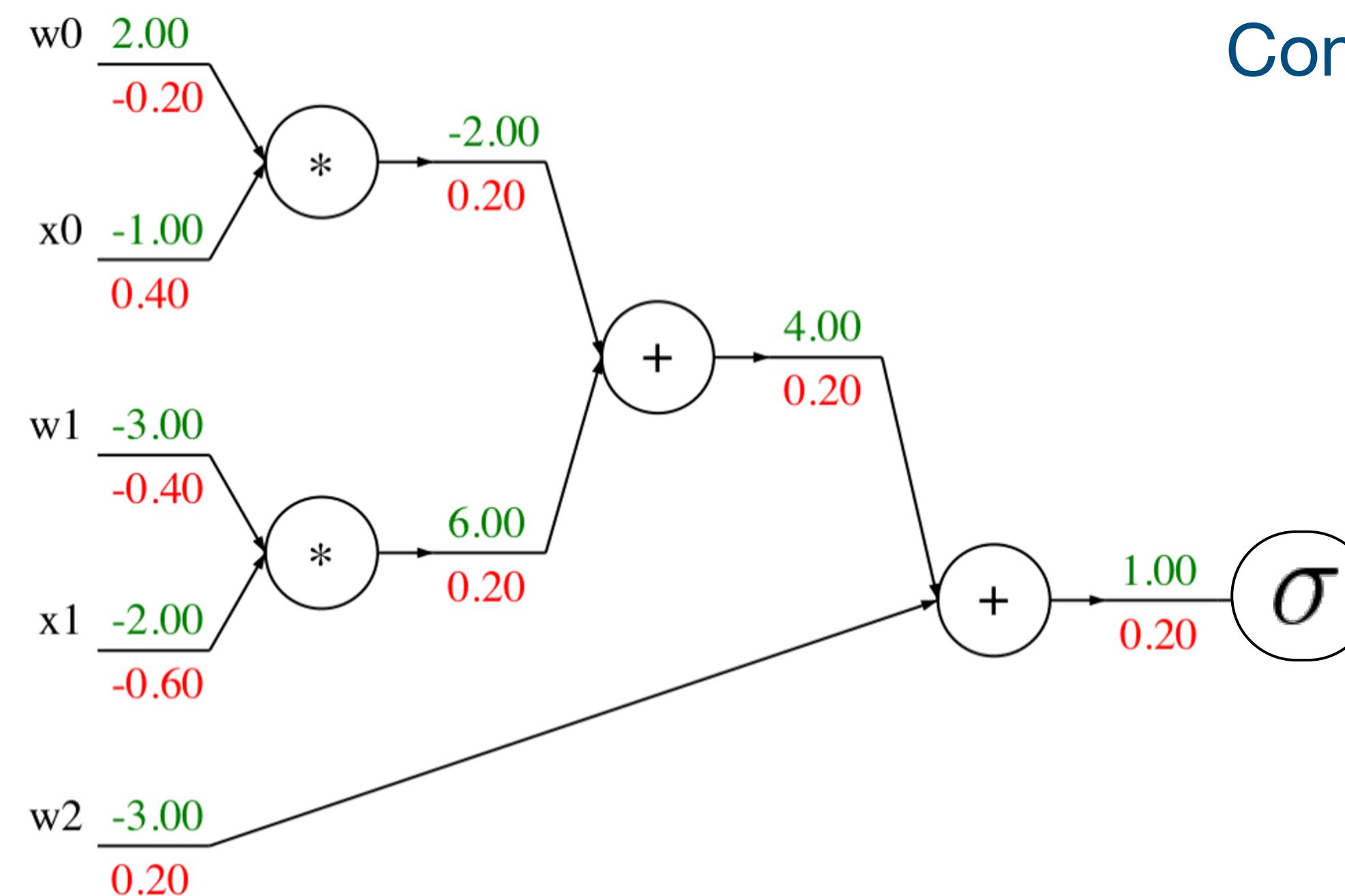
Backprop Implementation: “Flat” gradient code



Forward pass:
Compute outputs

```
def f(w0, x0, w1, x1, w2):  
    s0 = w0 * x0  
    s1 = w1 * x1  
    s2 = s0 + s1  
    s3 = s2 + w2  
    L = sigmoid(s3)
```

Backprop Implementation: “Flat” gradient code



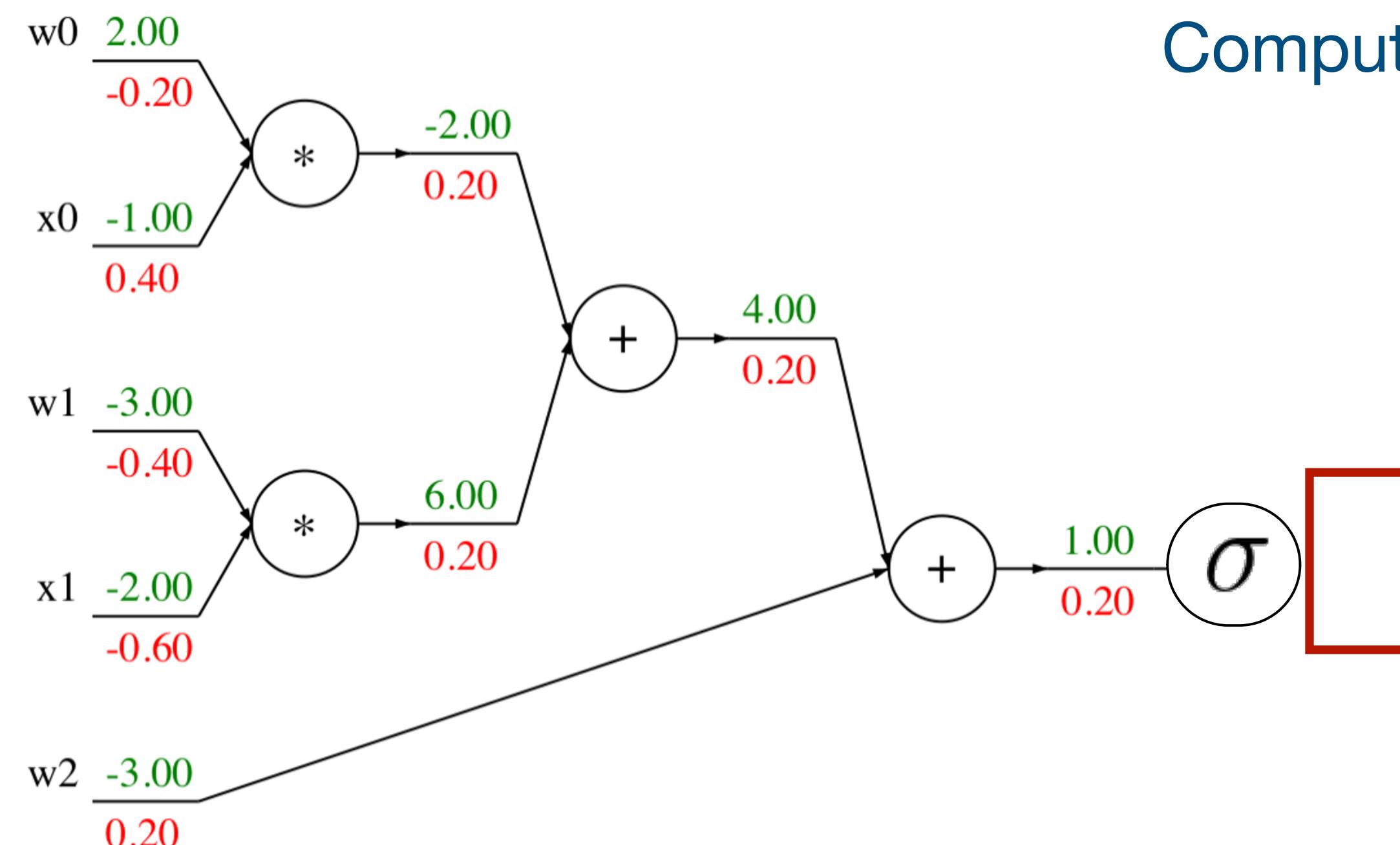
Forward pass:
Compute outputs

```
def f(w0, x0, w1, x1, w2):
    s0 = w0 * x0
    s1 = w1 * x1
    s2 = s0 + s1
    s3 = s2 + w2
    L = sigmoid(s3)
```

Backward pass:
Compute gradients

```
grad_L = 1.0
grad_s3 = grad_L * (1 - L) * L
grad_w2 = grad_s3
grad_s2 = grad_s3
grad_s0 = grad_s2
grad_s1 = grad_s2
grad_w1 = grad_s1 * x1
grad_x1 = grad_s1 * w1
grad_w0 = grad_s0 * x0
grad_x0 = grad_s0 * w0
```

Backprop Implementation: “Flat” gradient code



Forward pass:
Compute outputs

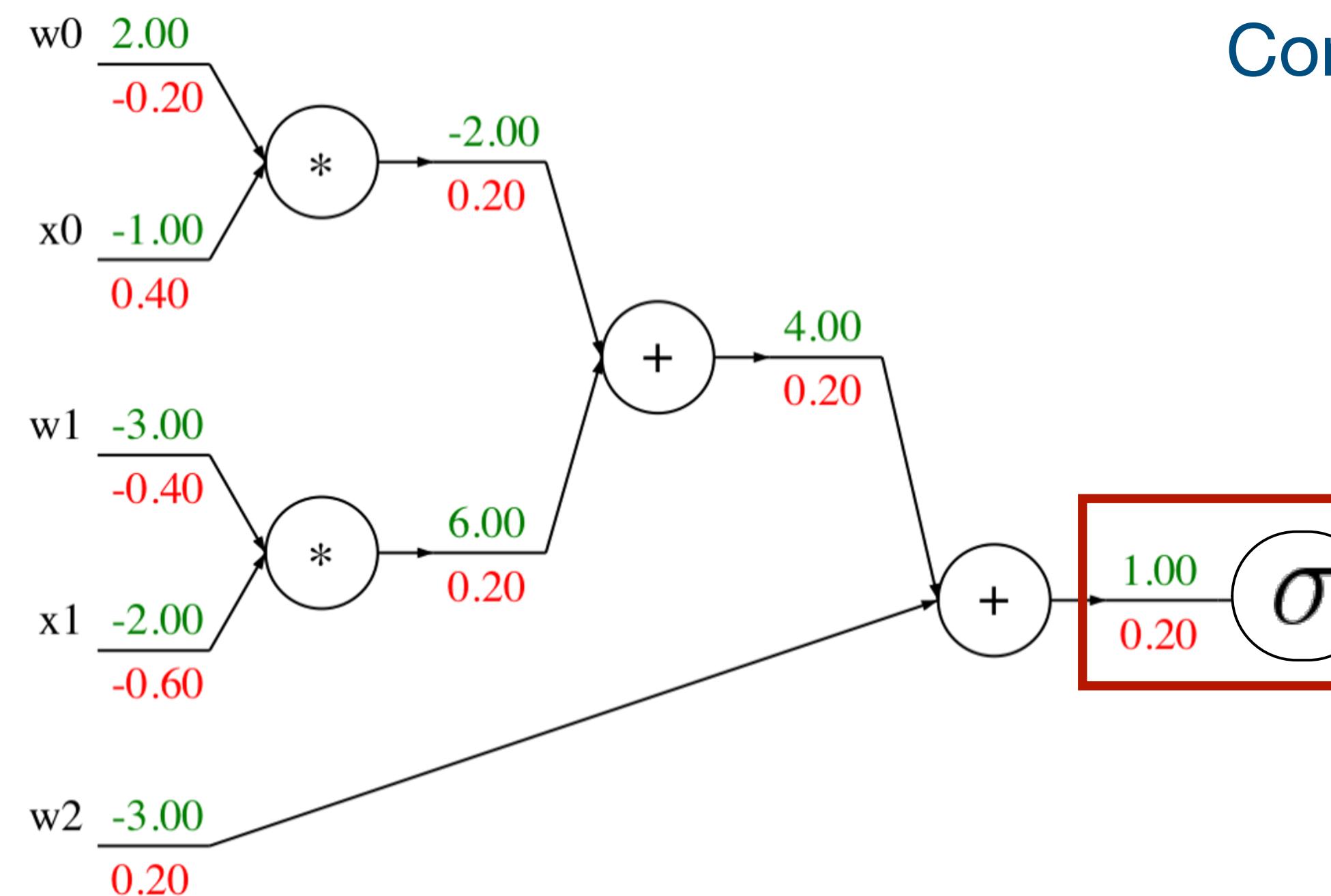
```
def f(w0, x0, w1, x1, w2):
    s0 = w0 * x0
    s1 = w1 * x1
    s2 = s0 + s1
    s3 = s2 + w2
    L = sigmoid(s3)
```

Base case

```
grad_L = 1.0
grad_s3 = grad_L * (1 - L) * L
grad_w2 = grad_s3
grad_s2 = grad_s3
grad_s0 = grad_s2
grad_s1 = grad_s2
grad_w1 = grad_s1 * x1
grad_x1 = grad_s1 * w1
grad_w0 = grad_s0 * x0
grad_x0 = grad_s0 * w0
```

Backward pass:
Compute gradients

Backprop Implementation: “Flat” gradient code



Forward pass:
Compute outputs

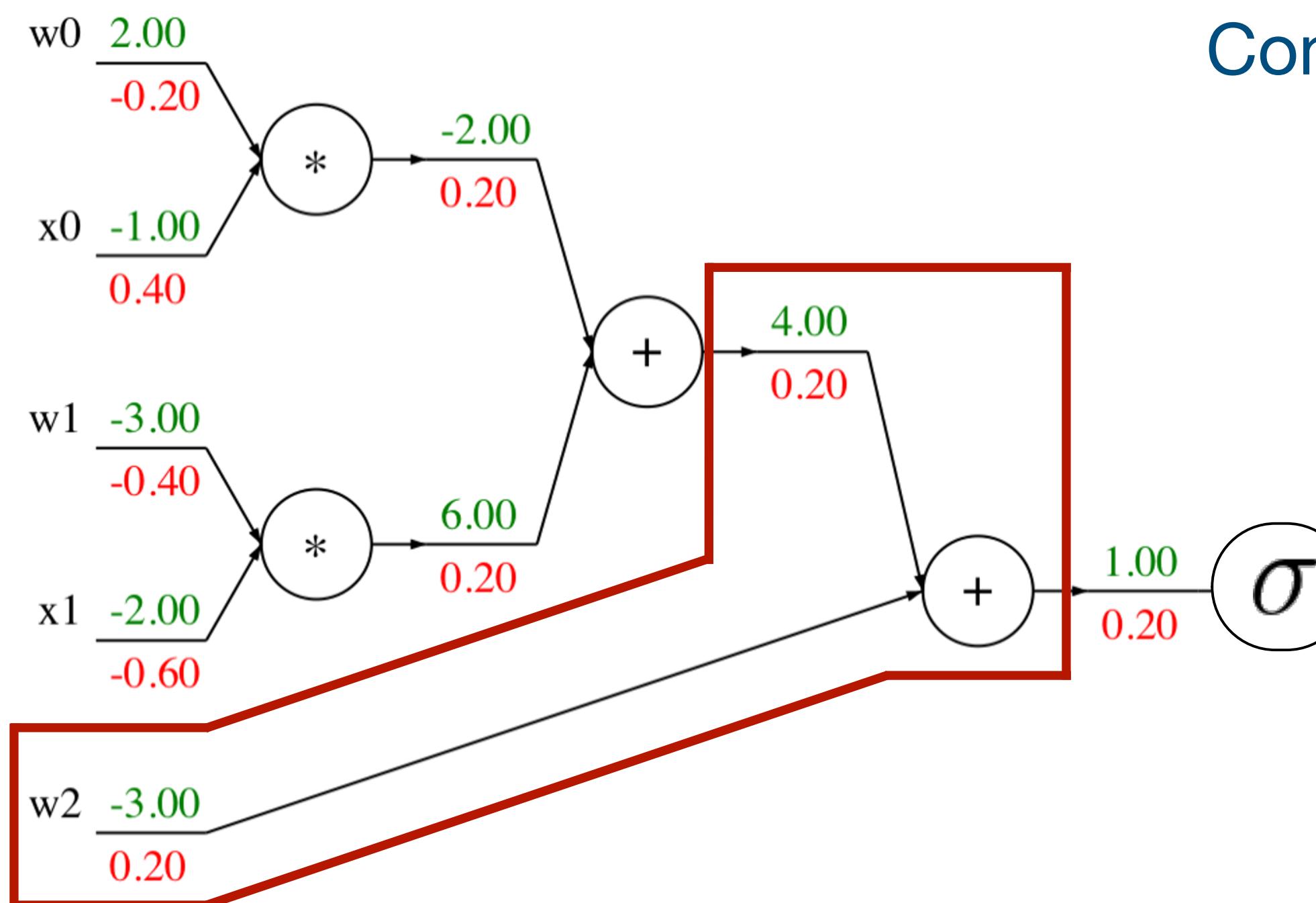
```
def f(w0, x0, w1, x1, w2):
    s0 = w0 * x0
    s1 = w1 * x1
    s2 = s0 + s1
    s3 = s2 + w2
    L = sigmoid(s3)
```

Sigmoid

```
grad_L = 1.0
grad_s3 = grad_L * (1 - L) * L
grad_w2 = grad_s3
grad_s2 = grad_s3
grad_s0 = grad_s2
grad_s1 = grad_s2
grad_w1 = grad_s1 * x1
grad_x1 = grad_s1 * w1
grad_w0 = grad_s0 * x0
grad_x0 = grad_s0 * w0
```

Backward pass:
Compute gradients

Backprop Implementation: “Flat” gradient code



Forward pass:
Compute outputs

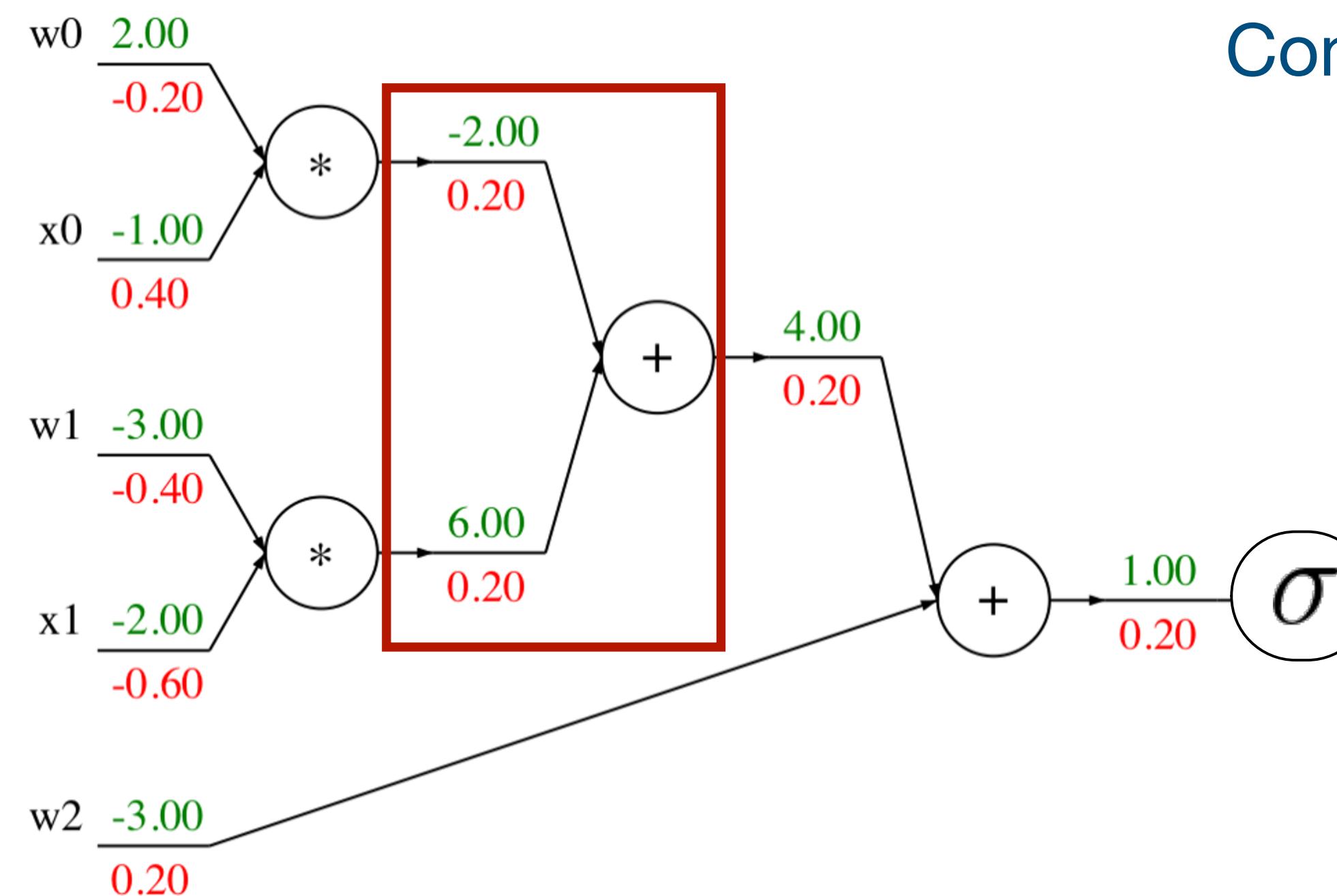
```
def f(w0, x0, w1, x1, w2):
    s0 = w0 * x0
    s1 = w1 * x1
    s2 = s0 + s1
    s3 = s2 + w2
    L = sigmoid(s3)
```

Add

```
grad_L = 1.0
grad_s3 = grad_L * (1 - L) * L
grad_w2 = grad_s3
grad_s2 = grad_s3
grad_s0 = grad_s2
grad_s1 = grad_s2
grad_w1 = grad_s1 * x1
grad_x1 = grad_s1 * w1
grad_w0 = grad_s0 * x0
grad_x0 = grad_s0 * w0
```

Backward pass:
Compute gradients

Backprop Implementation: “Flat” gradient code



Forward pass:
Compute outputs

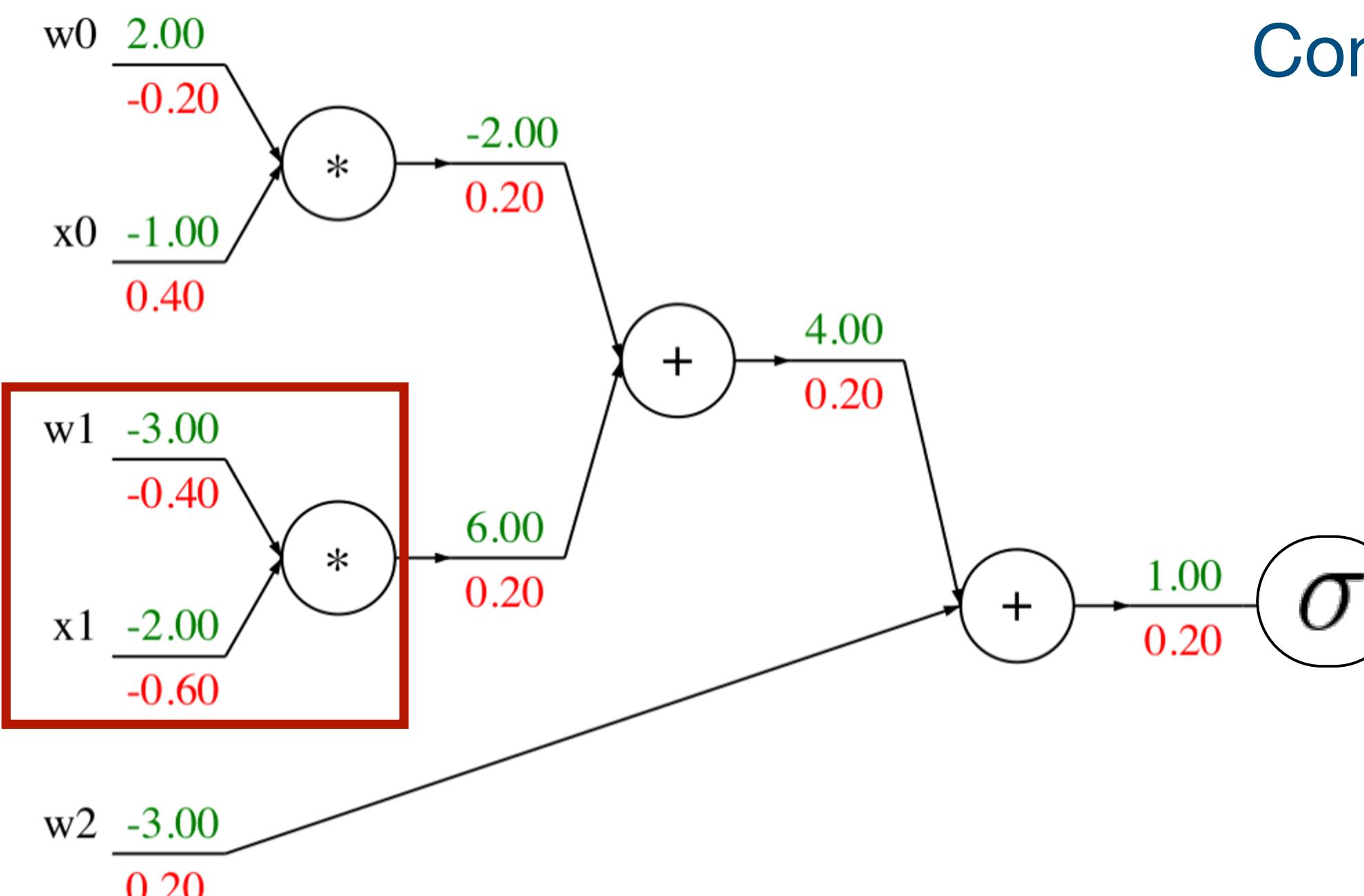
```
def f(w0, x0, w1, x1, w2):
    s0 = w0 * x0
    s1 = w1 * x1
    s2 = s0 + s1
    s3 = s2 + w2
    L = sigmoid(s3)

    grad_L = 1.0
    grad_s3 = grad_L * (1 - L) * L
    grad_w2 = grad_s3
    grad_s2 = grad_s3
    grad_s0 = grad_s2
    grad_s1 = grad_s2
    grad_w1 = grad_s1 * x1
    grad_x1 = grad_s1 * w1
    grad_w0 = grad_s0 * x0
    grad_x0 = grad_s0 * w0
```

Add

Backward pass:
Compute gradients

Backprop Implementation: “Flat” gradient code



Forward pass:
Compute outputs

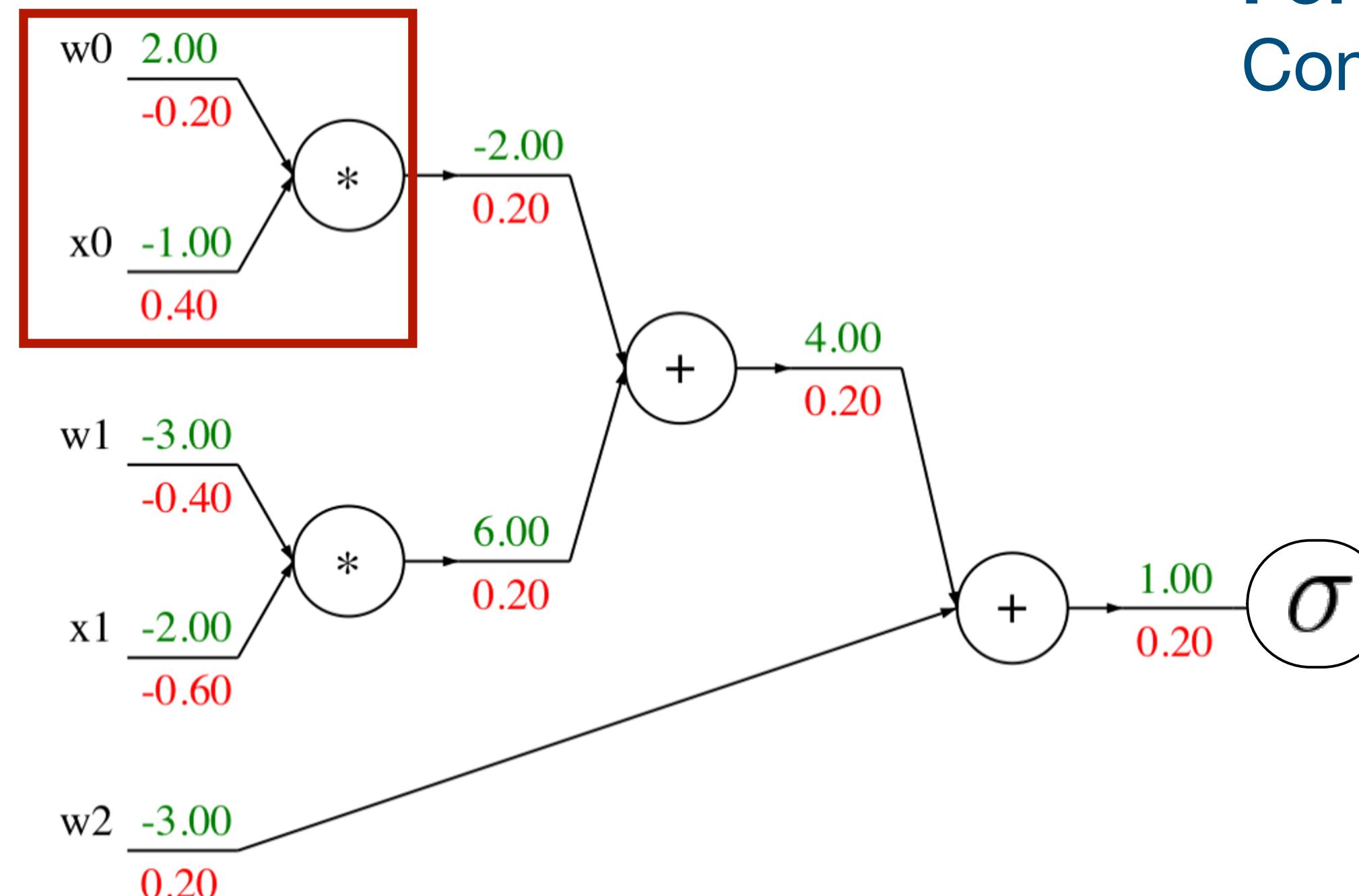
```
def f(w0, x0, w1, x1, w2):
    s0 = w0 * x0
    s1 = w1 * x1
    s2 = s0 + s1
    s3 = s2 + w2
    L = sigmoid(s3)

    grad_L = 1.0
    grad_s3 = grad_L * (1 - L) * L
    grad_w2 = grad_s3
    grad_s2 = grad_s3
    grad_s0 = grad_s2
    grad_s1 = grad_s2
    grad_w1 = grad_s1 * x1
    grad_x1 = grad_s1 * w1
    grad_w0 = grad_s0 * x0
    grad_x0 = grad_s0 * w0
```

Multiply

Backward pass:
Compute gradients

Backprop Implementation: “Flat” gradient code



Forward pass:
Compute outputs

```
def f(w0, x0, w1, x1, w2):
    s0 = w0 * x0
    s1 = w1 * x1
    s2 = s0 + s1
    s3 = s2 + w2
    L = sigmoid(s3)

    grad_L = 1.0
    grad_s3 = grad_L * (1 - L) * L
    grad_w2 = grad_s3
    grad_s2 = grad_s3
    grad_s0 = grad_s2
    grad_s1 = grad_s2
    grad_w1 = grad_s1 * x1
    grad_x1 = grad_s1 * w1
    grad_w0 = grad_s0 * x0
    grad_x0 = grad_s0 * w0
```

Multiply

Backward pass:
Compute gradients

“Flat” Backprop: Do this for Project 1 & 2

Forward pass: Compute outputs

```
#####
# TODO:                                     #
# Implement a vectorized version of the structured SVM loss, storing the   #
# result in loss.                         #
#####
# Replace "pass" statement with your code
num_classes = W.shape[1]
num_train = X.shape[0]
score = # ...
correct_class_score = # ...
margin = # ...
data_loss = # ...
reg_loss = # ...
loss += data_loss + reg_loss
#####
#                                         END OF YOUR CODE
#####
#
```

Backward pass: Compute gradients

```
#####
# TODO:                                     #
# Implement a vectorized version of the gradient for the structured SVM    #
# loss, storing the result in dW.                                         #
#
# Hint: Instead of computing the gradient from scratch, it may be easier   #
# to reuse some of the intermediate values that you used to compute the    #
# loss.
#####
# Replace "pass" statement with your code
dmargins = # ...
dscores = # ...
dW = # ...
#####
#                                         END OF YOUR CODE
#####
#
```



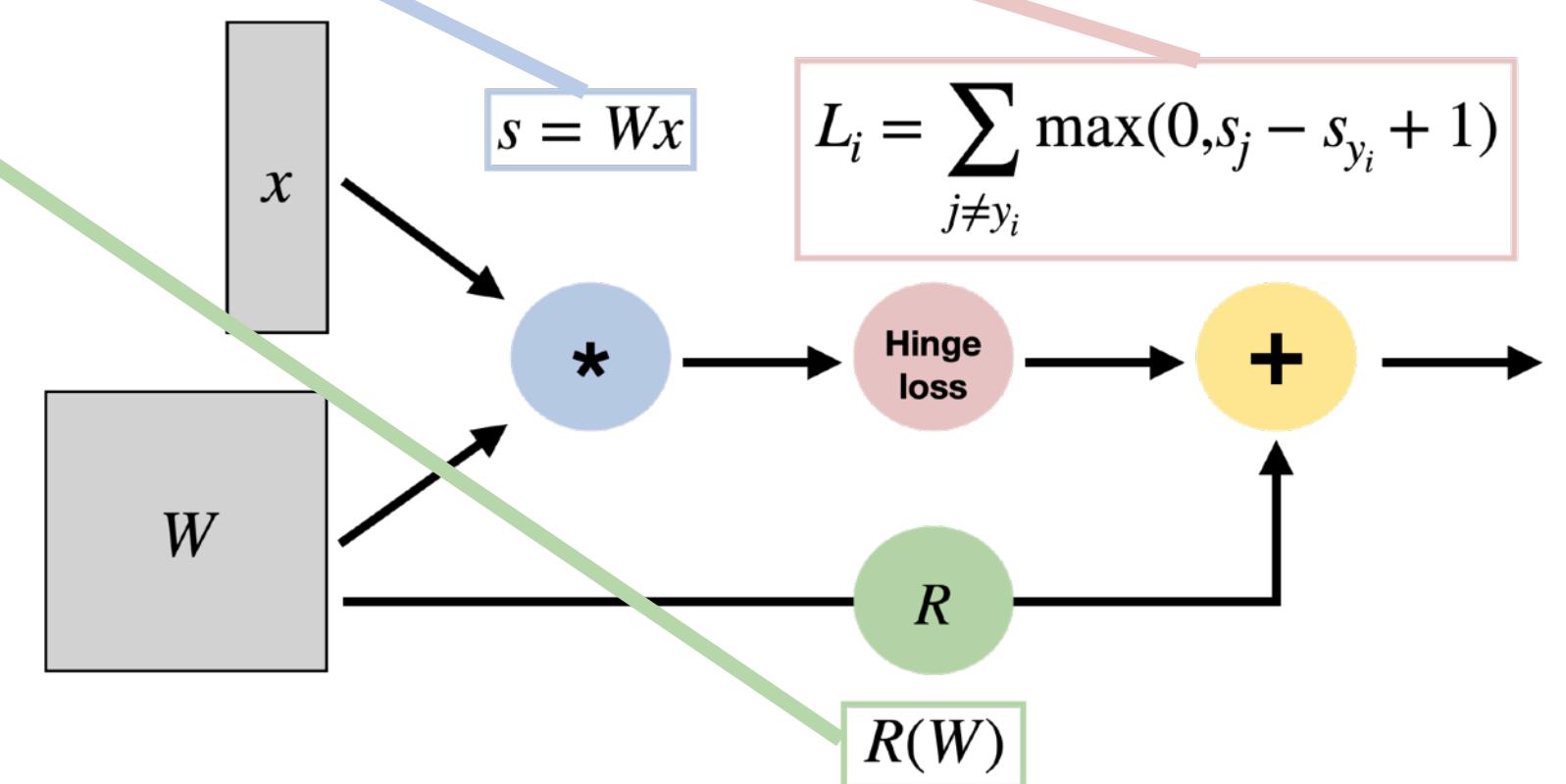
“Flat” Backprop: Do this for Project 1 & 2

Forward pass: Compute outputs

```
#####
# TODO: #
# Implement a vectorized version of the structured SVM loss, storing the #
# result in loss. #
#####
# Replace "pass" statement with your code
num_classes = W.shape[1]
num_train = X.shape[0]
score = # ...
correct_class_score = # ...
margin = # ...
data_loss = # ...
reg_loss = # ...
loss += data_loss + reg_loss
#####
# END OF YOUR CODE
#####
#
```

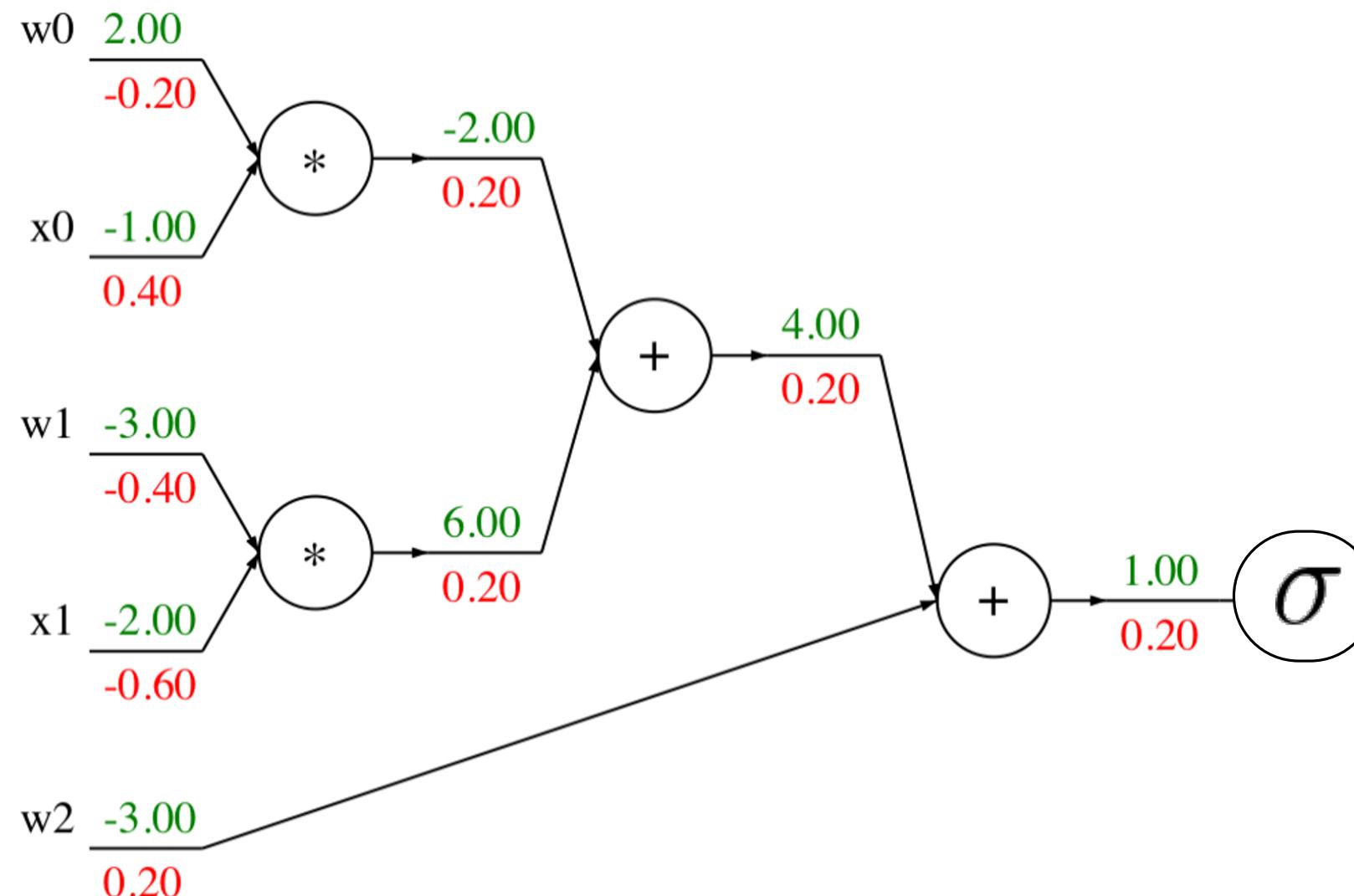
Backward pass: Compute gradients

```
#####
# TODO: #
# Implement a vectorized version of the gradient for the structured SVM #
# loss, storing the result in dW. #
#
# Hint: Instead of computing the gradient from scratch, it may be easier #
# to reuse some of the intermediate values that you used to compute the #
# loss.
#####
# Replace "pass" statement with your code
dmargins = # ...
dscores = # ...
dW = # ...
#####
# END OF YOUR CODE
#####
#
```



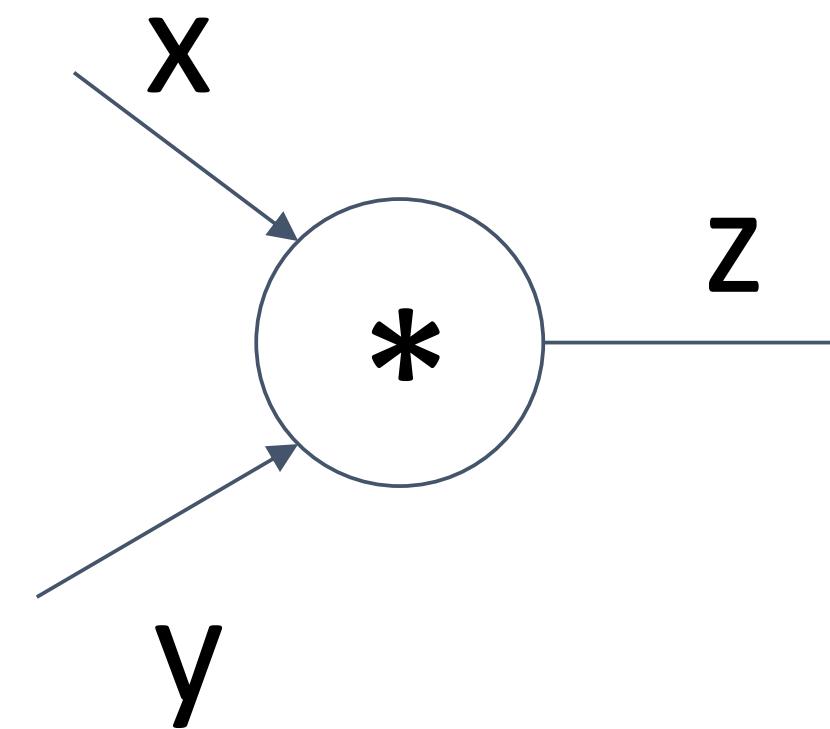
Backprop Implementation: Modular API

Graph (or Net) object (*rough pseudo code*)



```
class ComputationalGraph(object):
    ...
    def forward(inputs):
        # 1. [pass inputs to input gates...]
        # 2. forward the computational graph:
        for gate in self.graph.nodes_topologically_sorted():
            gate.forward()
        return loss # the final gate in the graph outputs the loss
    def backward():
        for gate in reversed(self.graph.nodes_topologically_sorted()):
            gate.backward() # little piece of backprop (chain rule applied)
        return inputs_gradients
```

Example: PyTorch Autograd Functions



(x, y, z are scalars)

```
class Multiply(torch.autograd.Function):
    @staticmethod
    def forward(ctx, x, y):
        ctx.save_for_backward(x, y)
        z = x * y
        return z

    @staticmethod
    def backward(ctx, grad_z):
        x, y = ctx.saved_tensors
        grad_x = y * grad_z # dz/dx * dL/dz
        grad_y = x * grad_z # dz/dy * dL/dz
        return grad_x, grad_y
```

Need to stash some values for use in backward

Upstream gradient

Multiply upstream and local gradients

So far: backprop with scalars

What about vector-valued functions?

Recap: Vector Derivatives

$$x \in \mathbb{R}, y \in \mathbb{R}$$

Regular derivative:

$$\frac{\partial y}{\partial x} \in \mathbb{R}$$

If x changes by a small amount, how much will y change?

Recap: Vector Derivatives

$$x \in \mathbb{R}, y \in \mathbb{R}$$

Regular derivative:

$$\frac{\partial y}{\partial x} \in \mathbb{R}$$

If x changes by a small amount, how much will y change?

$$x \in \mathbb{R}^N, y \in \mathbb{R}$$

Derivative is **Gradient**:

$$\begin{aligned}\frac{\partial y}{\partial x} &\in \mathbb{R}^N, \\ \left(\frac{\partial y}{\partial x}\right)_i &= \frac{\partial y}{\partial x_i}\end{aligned}$$

For each element of x , if it changes by a small amount then how much will y change?



Recap: Vector Derivatives

$$x \in \mathbb{R}, y \in \mathbb{R}$$

Regular derivative:

$$\frac{\partial y}{\partial x} \in \mathbb{R}$$

If x changes by a small amount, how much will y change?

$$x \in \mathbb{R}^N, y \in \mathbb{R}$$

Derivative is **Gradient**:

$$\begin{aligned}\frac{\partial y}{\partial x} &\in \mathbb{R}^N, \\ \left(\frac{\partial y}{\partial x}\right)_i &= \frac{\partial y}{\partial x_i}\end{aligned}$$

For each element of x , if it changes by a small amount then how much will y change?

$$x \in \mathbb{R}^N, y \in \mathbb{R}^M$$

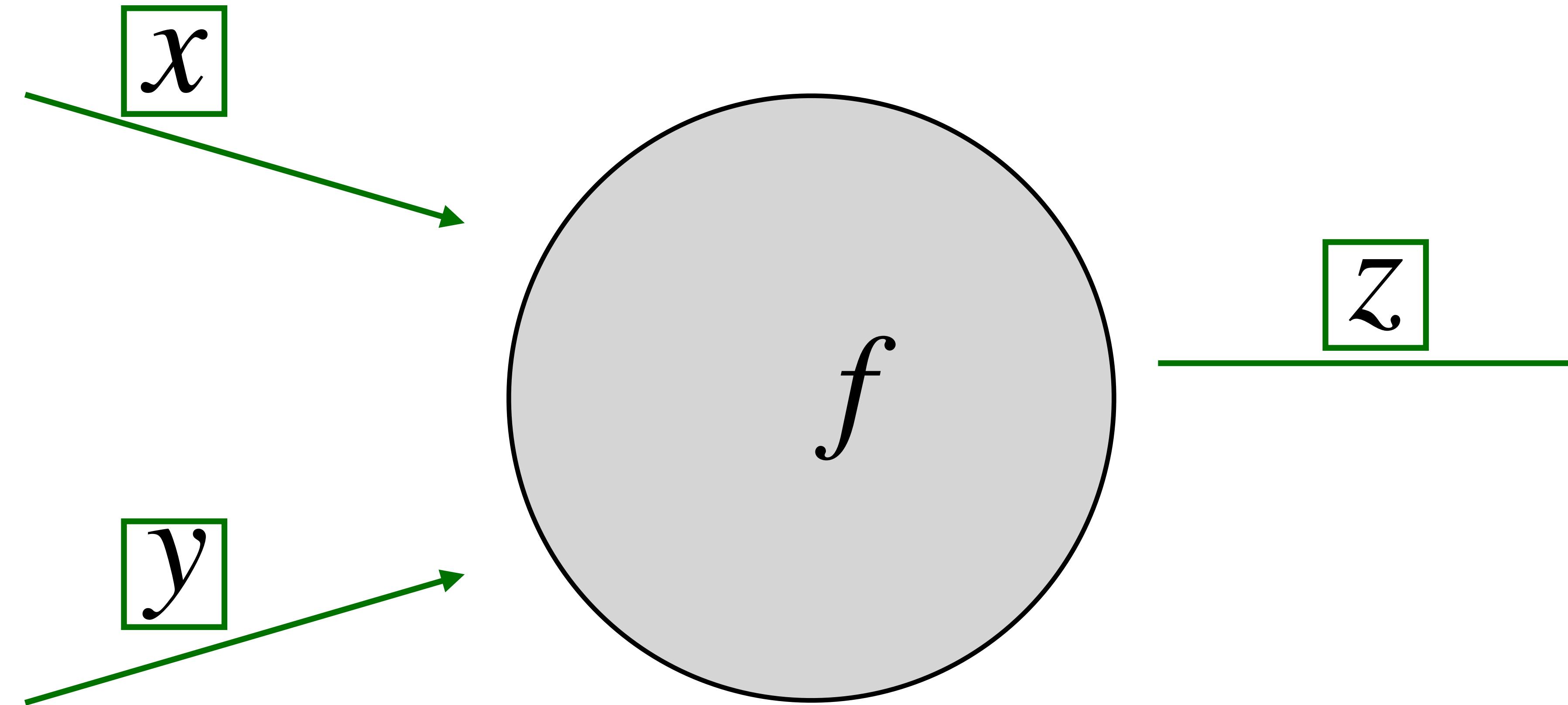
Derivative is **Jacobian**:

$$\begin{aligned}\frac{\partial y}{\partial x} &\in \mathbb{R}^{N \times M} \\ \left(\frac{\partial y}{\partial x}\right)_{i,j} &= \frac{\partial y_j}{\partial x_i}\end{aligned}$$

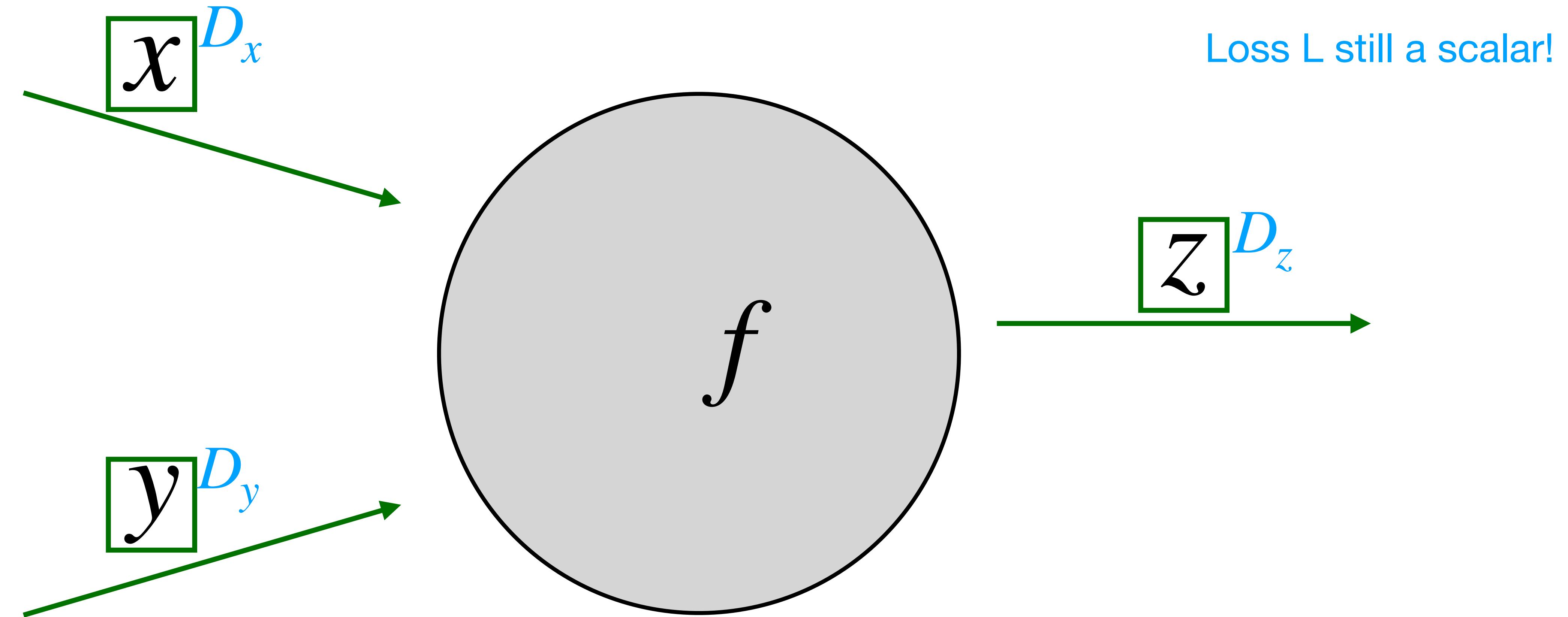
For each element of x , if it changes by a small amount then how much will each element of y change?



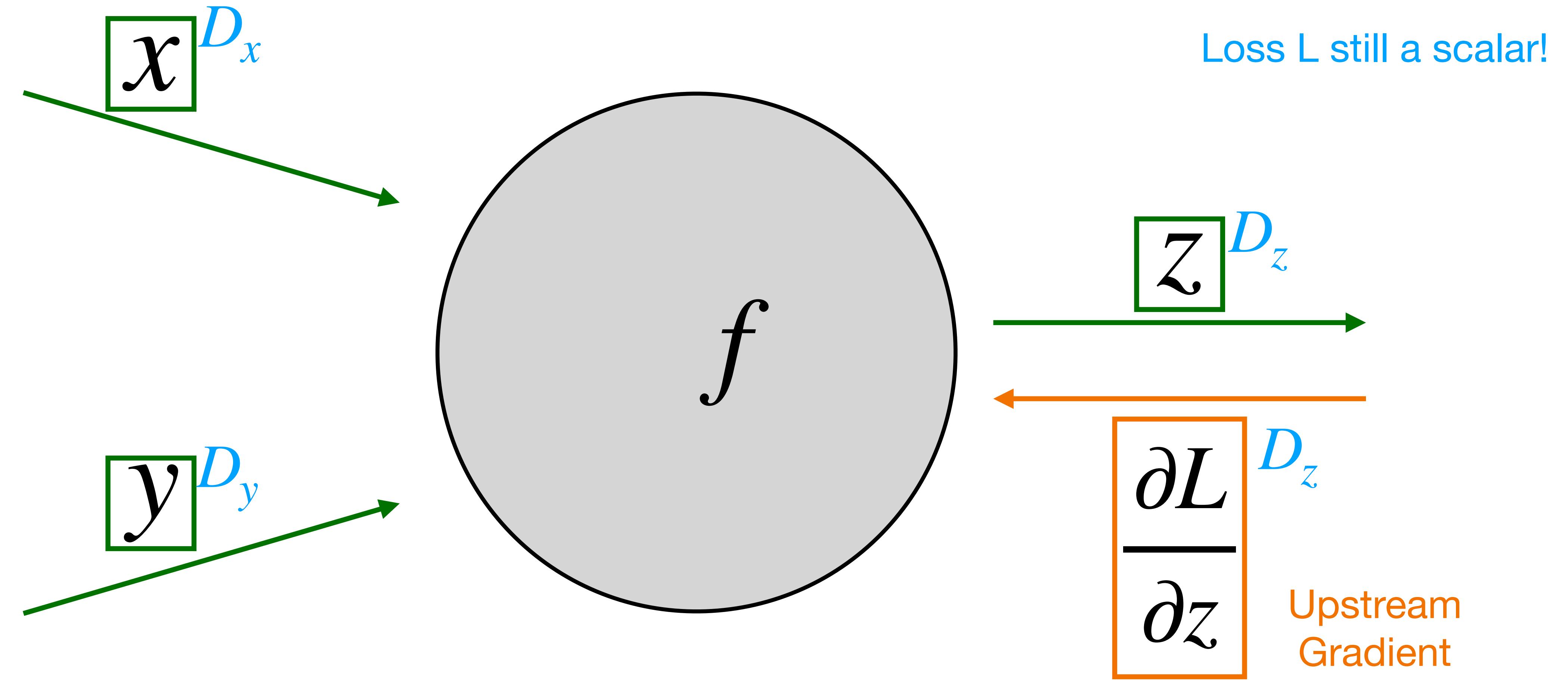
Backprop with Vectors



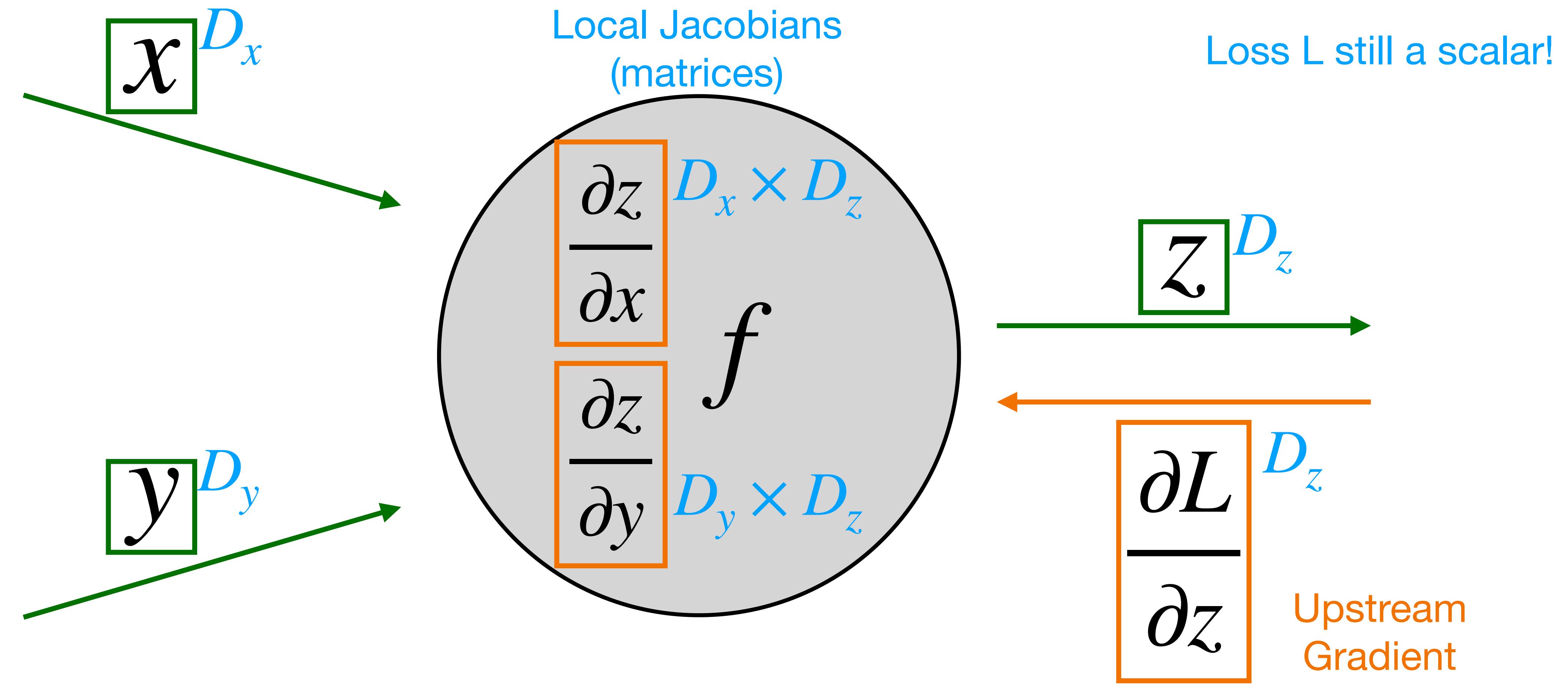
Backprop with Vectors



Backprop with Vectors

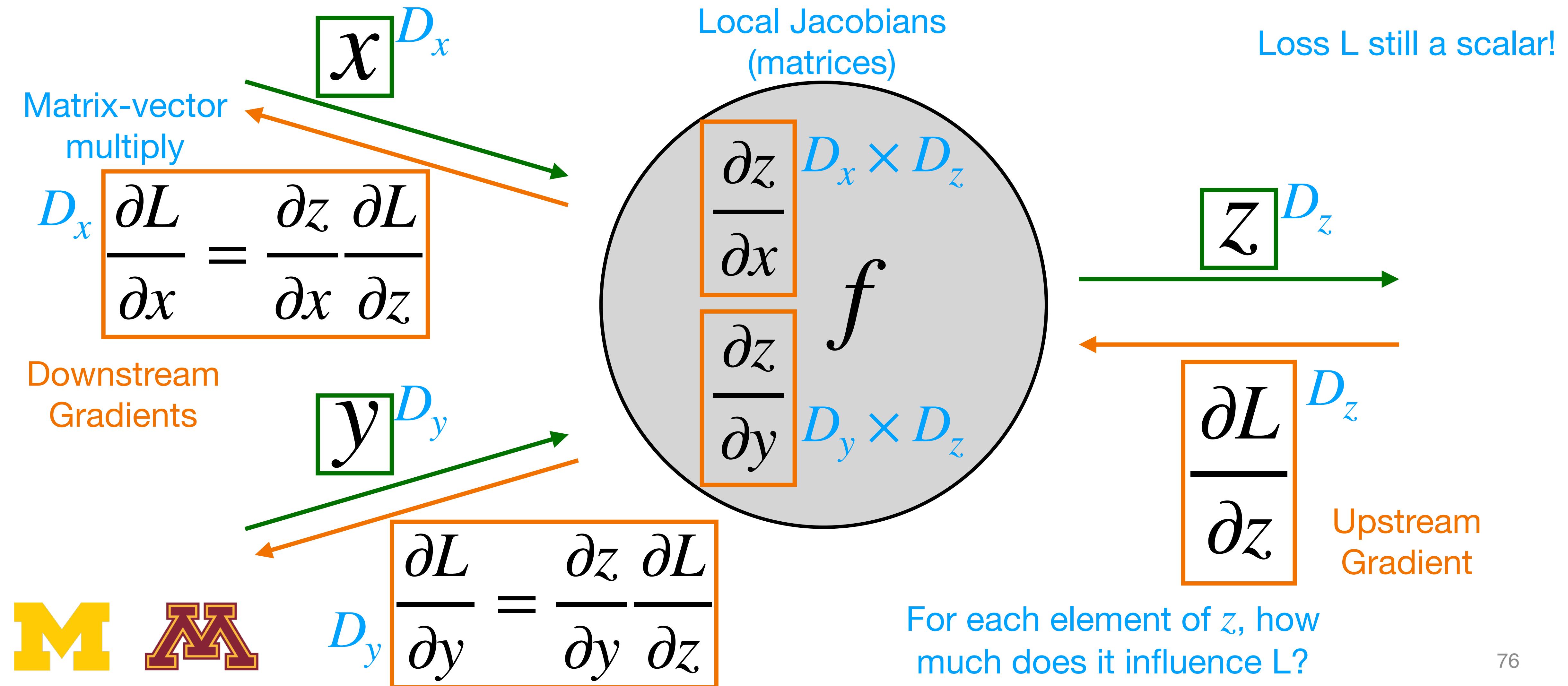


Backprop with Vectors



For each element of z , how
much does it influence L ?

Backprop with Vectors



Backprop with Vectors

4D input x:

$$\begin{bmatrix} 1 \\ -2 \\ 3 \\ -1 \end{bmatrix} \longrightarrow$$

$$f(x) = \max(0, x)$$

(elementwise)

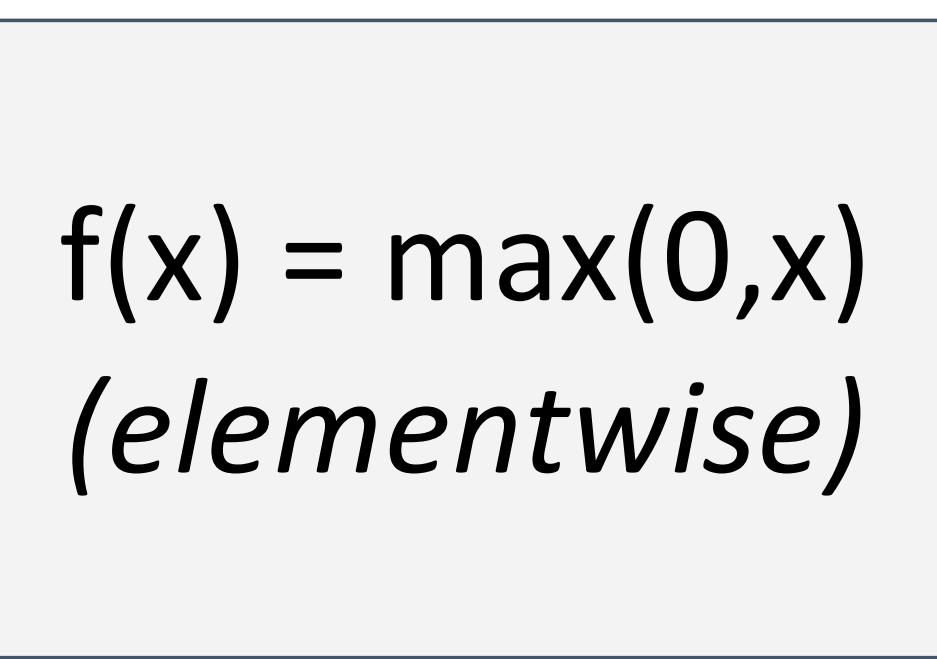
4D output y:

$$\begin{array}{l} \longrightarrow [1] \\ \longrightarrow [0] \\ \longrightarrow [3] \\ \longrightarrow [0] \end{array}$$

Backprop with Vectors

4D input x:

$$\begin{bmatrix} 1 \\ -2 \\ 3 \\ -1 \end{bmatrix} \longrightarrow$$



4D output y:

$$\longrightarrow \begin{bmatrix} 1 \\ 0 \\ 3 \\ 0 \end{bmatrix}$$

4D dL/dy :

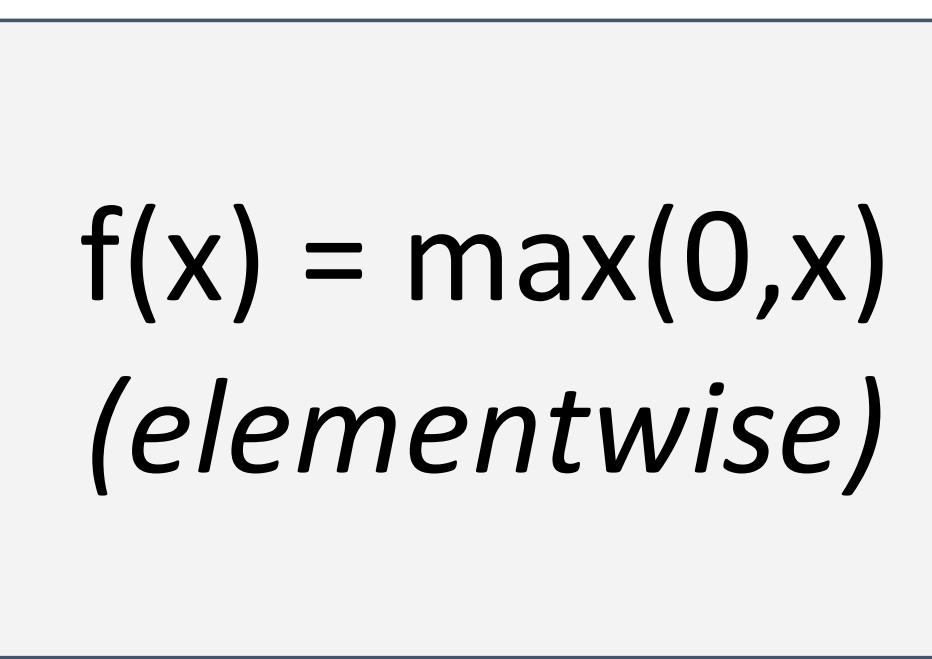
$$\begin{array}{c} \longleftarrow [4] \longrightarrow \\ \longleftarrow [-1] \longrightarrow \\ \longleftarrow [5] \longrightarrow \\ \longleftarrow [9] \longrightarrow \end{array}$$

Upstream
gradient

Backprop with Vectors

4D input x:

$$\begin{bmatrix} 1 \\ -2 \\ 3 \\ -1 \end{bmatrix} \longrightarrow$$



4D output y:

$$\longrightarrow \begin{bmatrix} 1 \\ 0 \\ 3 \\ 0 \end{bmatrix}$$

$[dy/dx] [dL/dy]$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 4 \\ -1 \\ 5 \\ 9 \end{bmatrix}$$

4D dL/dy :

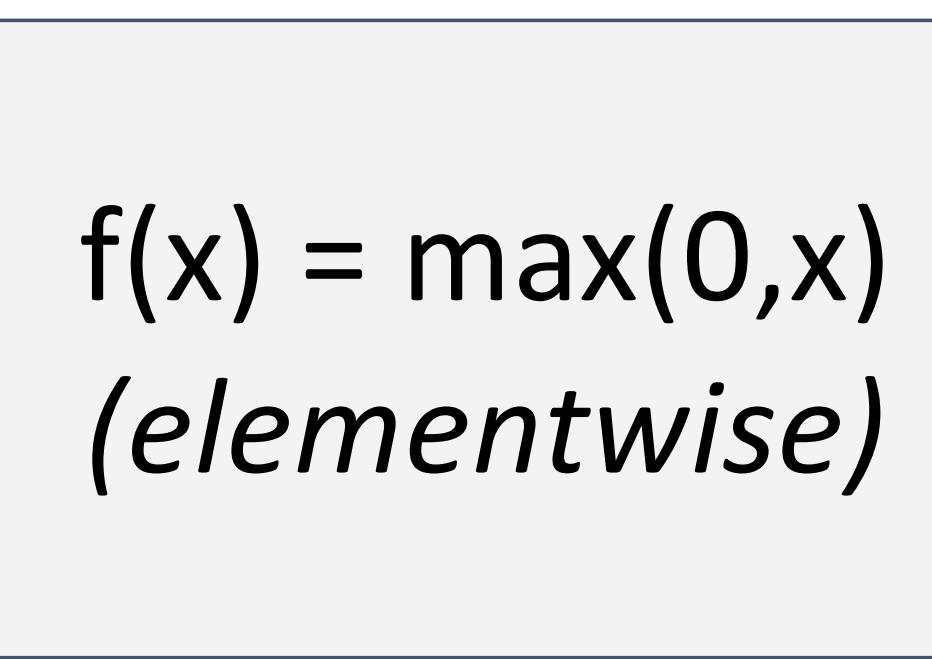
$$\begin{bmatrix} 4 \\ -1 \\ 5 \\ 9 \end{bmatrix}$$

Upstream
gradient

Backprop with Vectors

4D input x:

$$\begin{bmatrix} 1 \\ -2 \\ 3 \\ -1 \end{bmatrix} \longrightarrow$$



4D output y:

$$\longrightarrow \begin{bmatrix} 1 \\ 0 \\ 3 \\ 0 \end{bmatrix}$$

4D dL/dx :

$[4]$	\longleftarrow	$[1 \ 0 \ 0 \ 0]$	$[4]$
$[0]$	\longleftarrow	$[0 \ 0 \ 0 \ 0]$	$[-1]$
$[5]$	\longleftarrow	$[0 \ 0 \ 1 \ 0]$	$[5]$
$[0]$	\longleftarrow	$[0 \ 0 \ 0 \ 0]$	$[9]$

4D dL/dy :

$[4]$	\longleftarrow	$[4]$
$[-1]$	\longleftarrow	$[-1]$
$[5]$	\longleftarrow	$[5]$
$[9]$	\longleftarrow	$[9]$

Upstream
gradient

Backprop with Vectors

4D input x:

$$\begin{bmatrix} 1 \\ -2 \\ 3 \\ -1 \end{bmatrix} \xrightarrow{\text{green arrow}}$$

$$f(x) = \max(0, x) \\ (\textit{elementwise})$$

4D output y:

$$\xrightarrow{\text{green arrow}} \begin{bmatrix} 1 \\ 0 \\ 3 \\ 0 \end{bmatrix}$$

Jacobian is **sparse**: off-diagonal entries all zero!
Never **explicitly** form Jacobian; instead use **implicit** multiplication

4D dL/dx :

$$\begin{bmatrix} 4 \\ 0 \\ 5 \\ 0 \end{bmatrix} \xleftarrow{\text{red arrow}} \begin{bmatrix} dy/dx \\ dL/dy \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 4 \\ -1 \\ 5 \\ 9 \end{bmatrix}$$

4D dL/dy :

$$\begin{bmatrix} 4 \\ -1 \\ 5 \\ 9 \end{bmatrix} \xleftarrow{\text{red arrow}} \begin{bmatrix} 4 \\ -1 \\ 5 \\ 9 \end{bmatrix} \quad \begin{array}{l} \xleftarrow{\text{red arrow}} \\ \xleftarrow{\text{red arrow}} \\ \xleftarrow{\text{red arrow}} \\ \xleftarrow{\text{red arrow}} \end{array} \quad \begin{array}{l} \xleftarrow{\text{red arrow}} \\ \xleftarrow{\text{red arrow}} \\ \xleftarrow{\text{red arrow}} \\ \xleftarrow{\text{red arrow}} \end{array} \quad \begin{array}{l} \text{Upstream} \\ \text{gradient} \end{array}$$

Backprop with Vectors

4D input x:

$$\begin{bmatrix} 1 \\ -2 \\ 3 \\ -1 \end{bmatrix} \xrightarrow{\quad} \begin{bmatrix} 1 \\ 0 \\ 3 \\ 0 \end{bmatrix}$$

$$f(x) = \max(0, x) \\ (\textit{elementwise})$$

4D output y:

$$\begin{bmatrix} 1 \\ 0 \\ 3 \\ 0 \end{bmatrix}$$

Jacobian is **sparse**: off-diagonal entries all zero!
Never **explicitly** form Jacobian; instead use **implicit** multiplication

4D dL/dx :

$$\begin{bmatrix} 4 \\ 0 \\ 5 \\ 0 \end{bmatrix} \leftarrow$$

$[dy/dx] [dL/dy]$

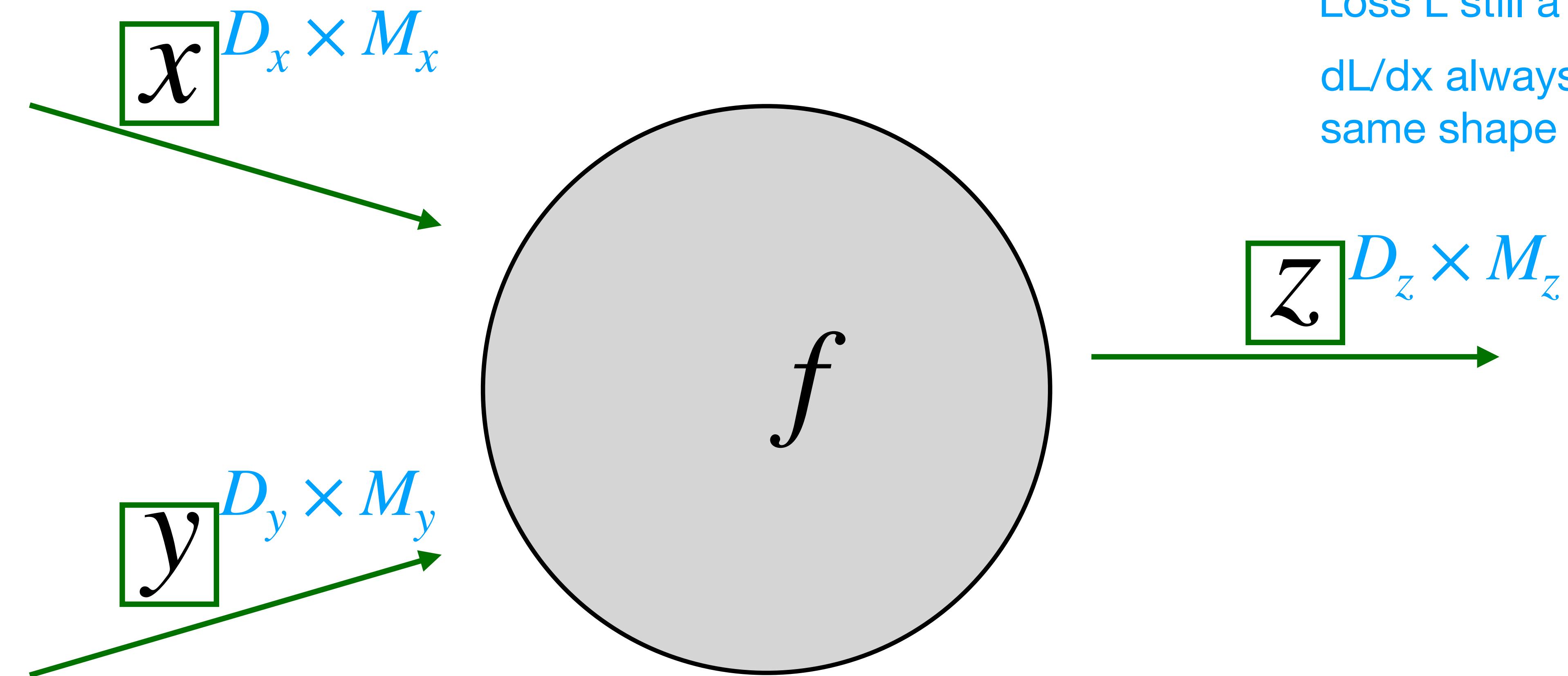
$$\left(\frac{\partial L}{\partial x}\right)_i = \begin{cases} \left(\frac{\partial L}{\partial y}\right)_i, & \text{if } x_i > 0 \\ 0, & \text{otherwise} \end{cases}$$

4D dL/dy :

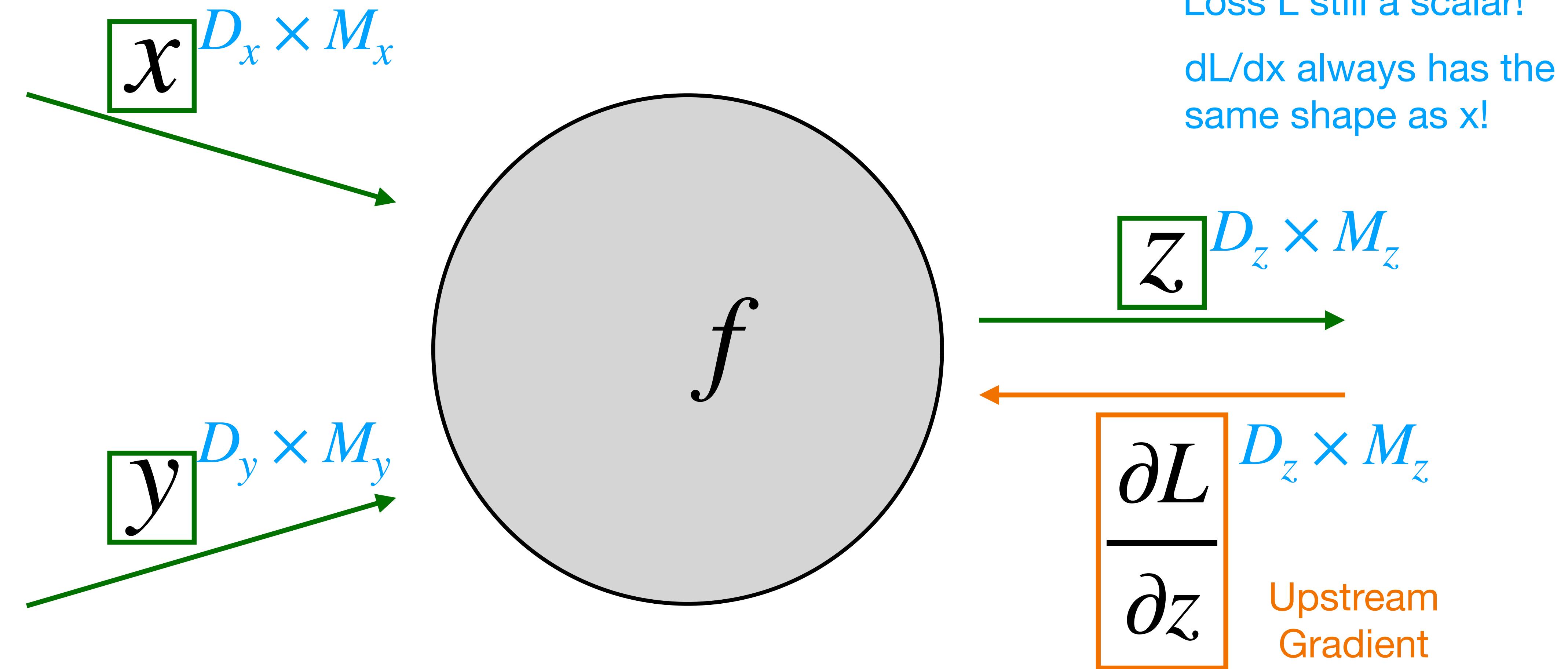
$$\begin{bmatrix} 4 \\ -1 \\ 5 \\ 9 \end{bmatrix} \leftarrow$$

Upstream
gradient

Backprop with Matrices (or Tensors)

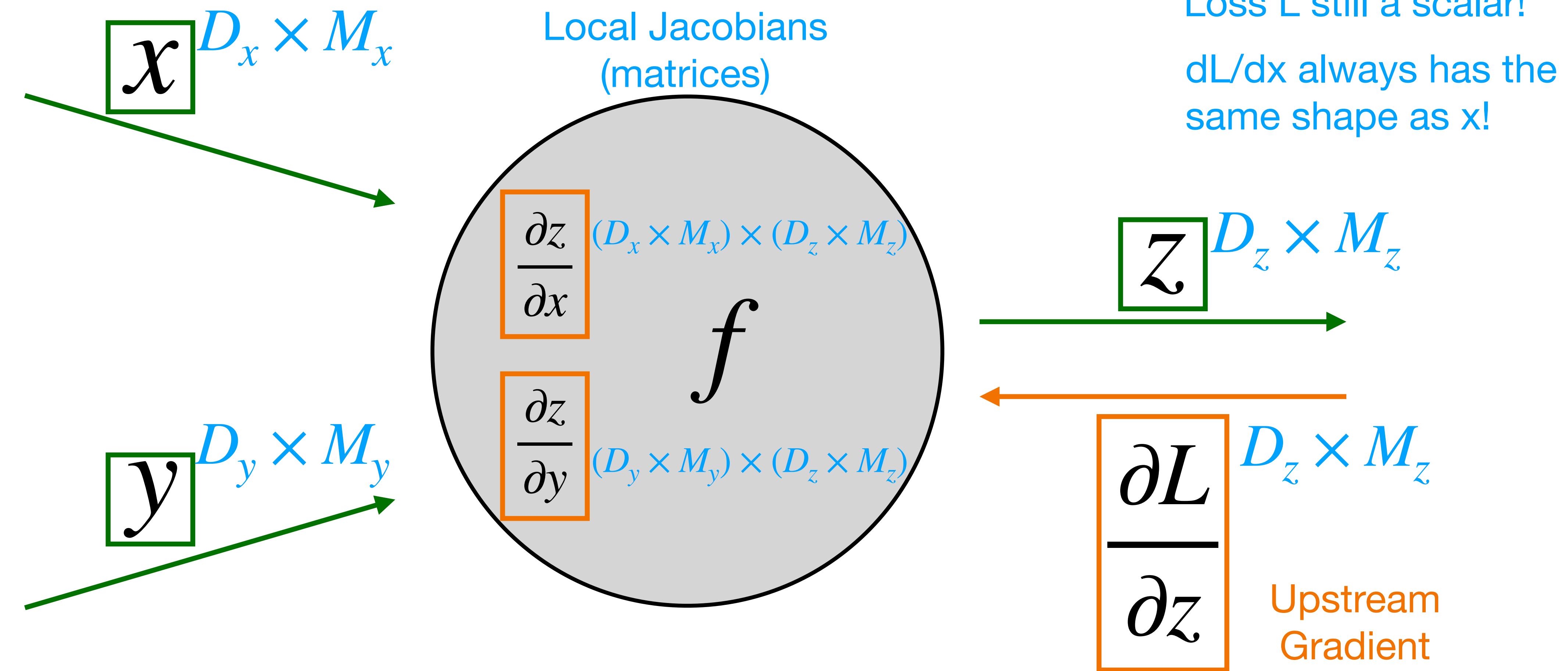


Backprop with Matrices (or Tensors)

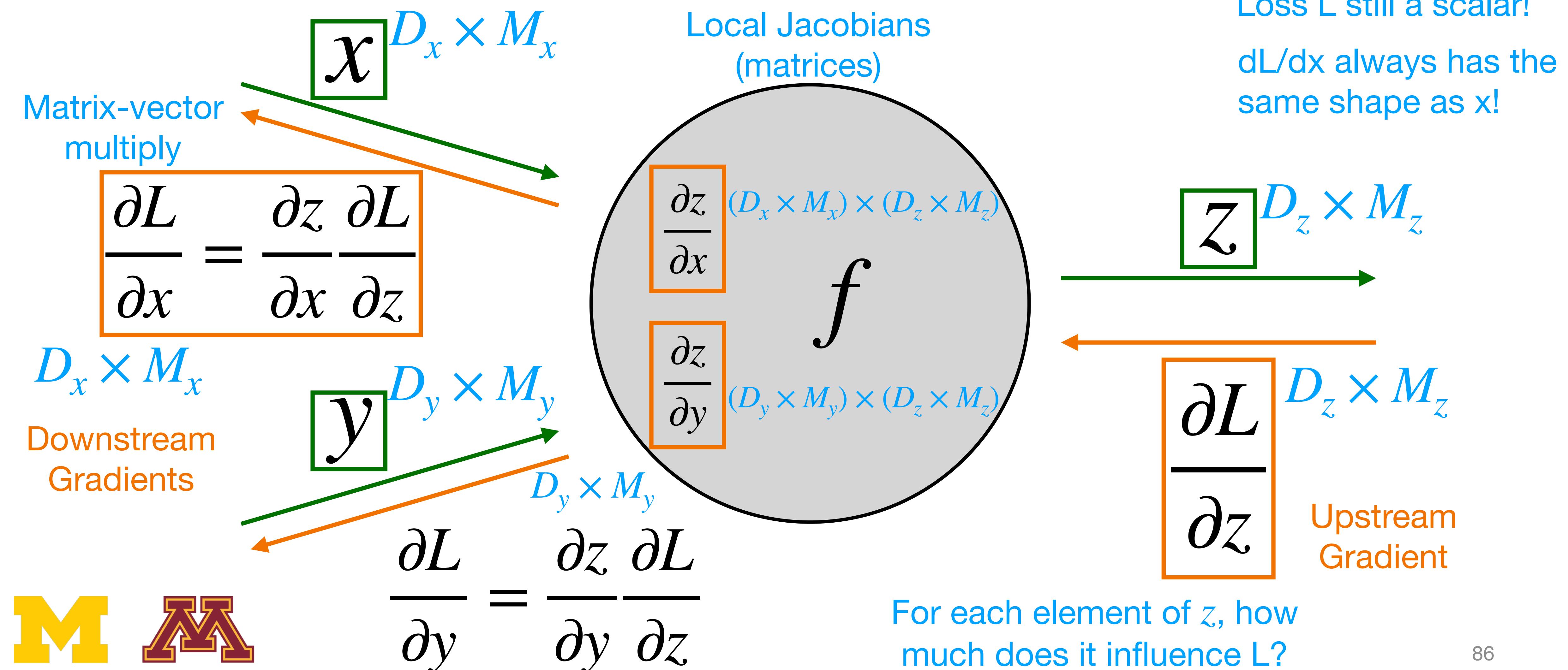


For each element of z , how much does it influence L ?

Backprop with Matrices (or Tensors)



Backprop with Matrices (or Tensors)



Example: Matrix Multiplication

$$\begin{array}{l} x: [N \times D] \\ \begin{bmatrix} 2 & 1 & -3 \\ -3 & 4 & 2 \end{bmatrix} \end{array} \quad \begin{array}{l} w: [D \times M] \\ \begin{bmatrix} 3 & 2 & 1 & -1 \\ 2 & 1 & 3 & 2 \\ 3 & 2 & 1 & -2 \end{bmatrix} \end{array}$$



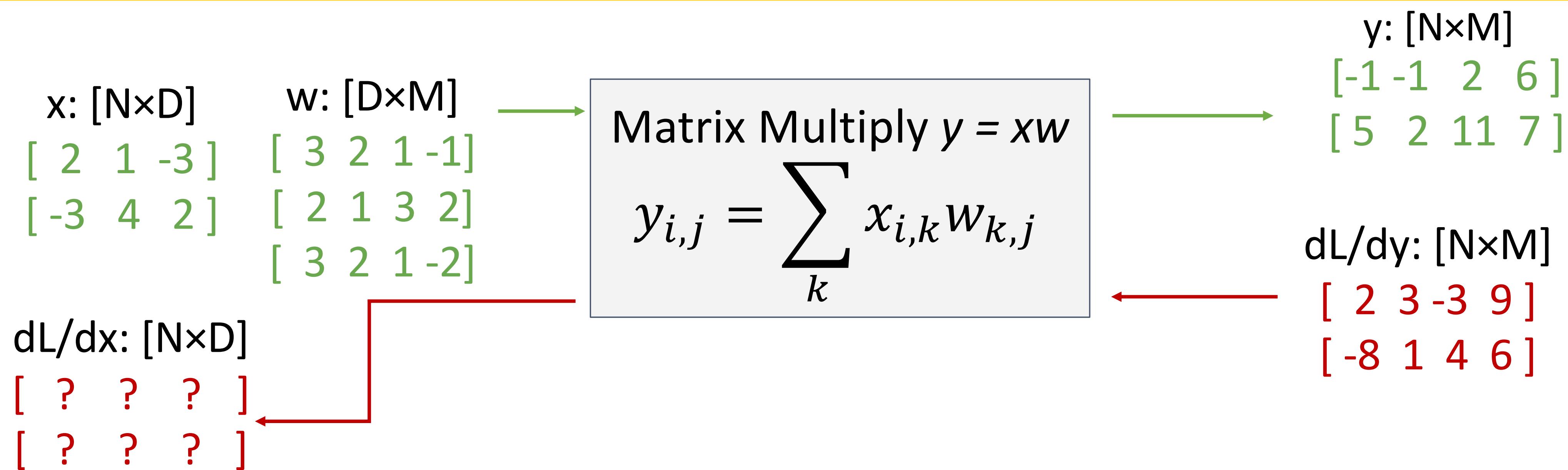
Matrix Multiply $y = xw$

$$y_{i,j} = \sum_k x_{i,k} w_{k,j}$$

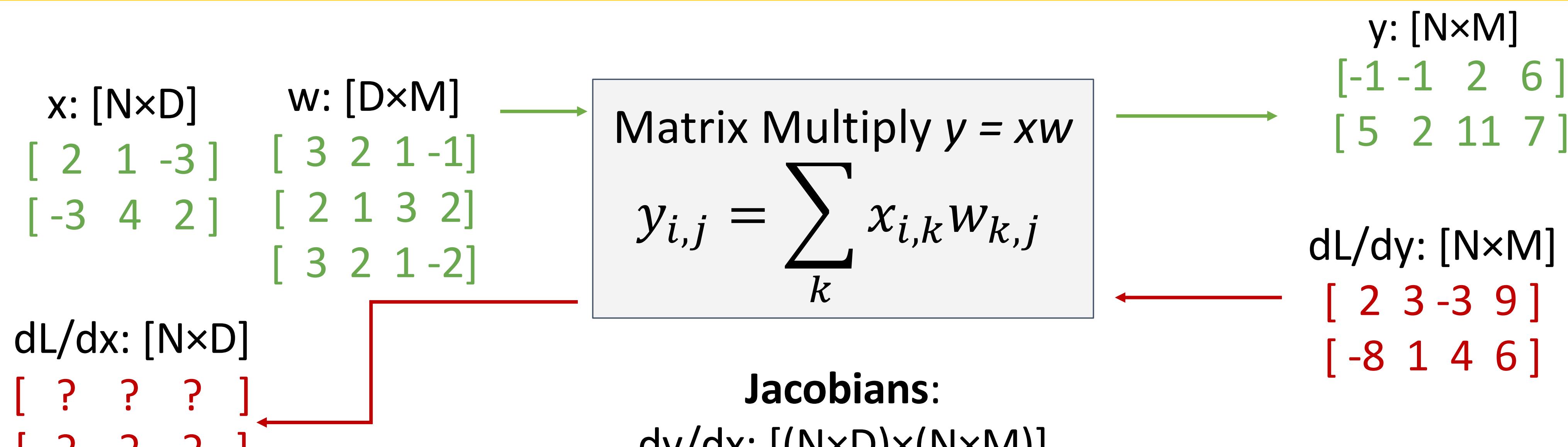


$$\begin{array}{l} y: [N \times M] \\ \begin{bmatrix} -1 & -1 & 2 & 6 \\ 5 & 2 & 11 & 7 \end{bmatrix} \end{array}$$

Example: Matrix Multiplication



Example: Matrix Multiplication

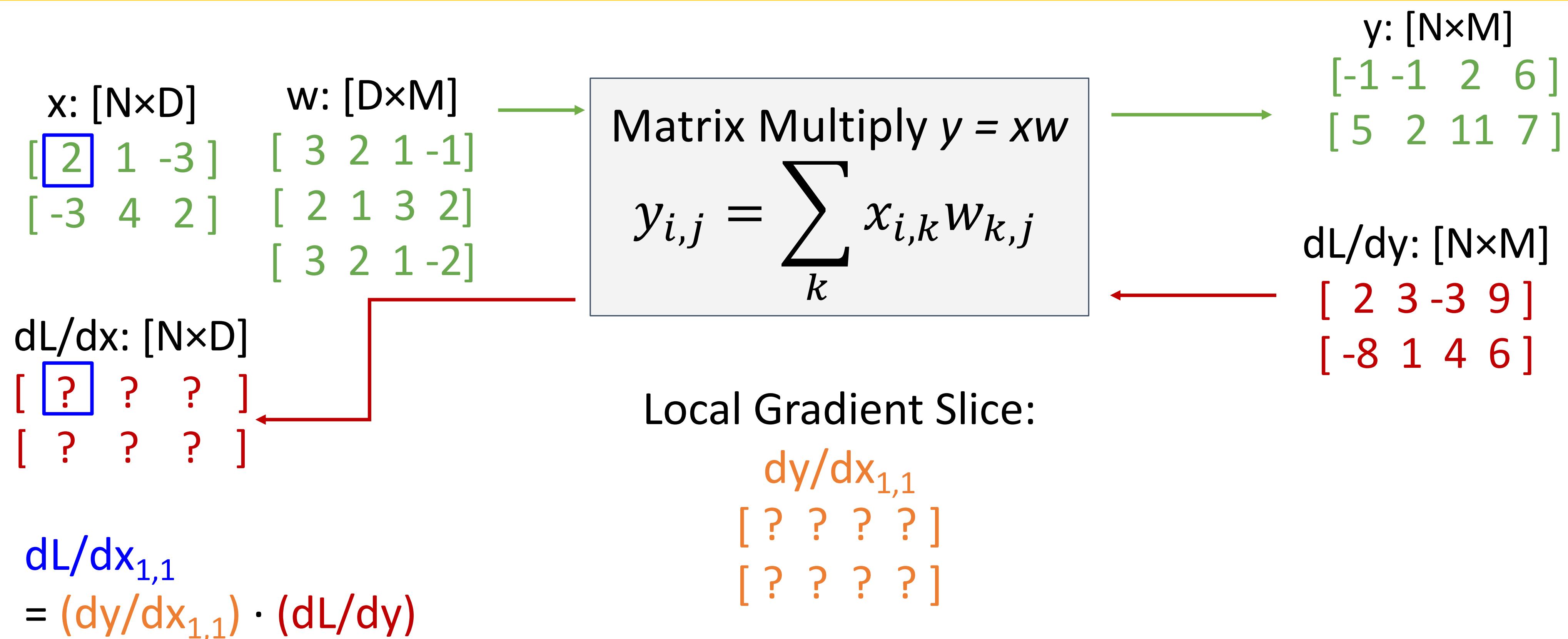


For a neural net we may have

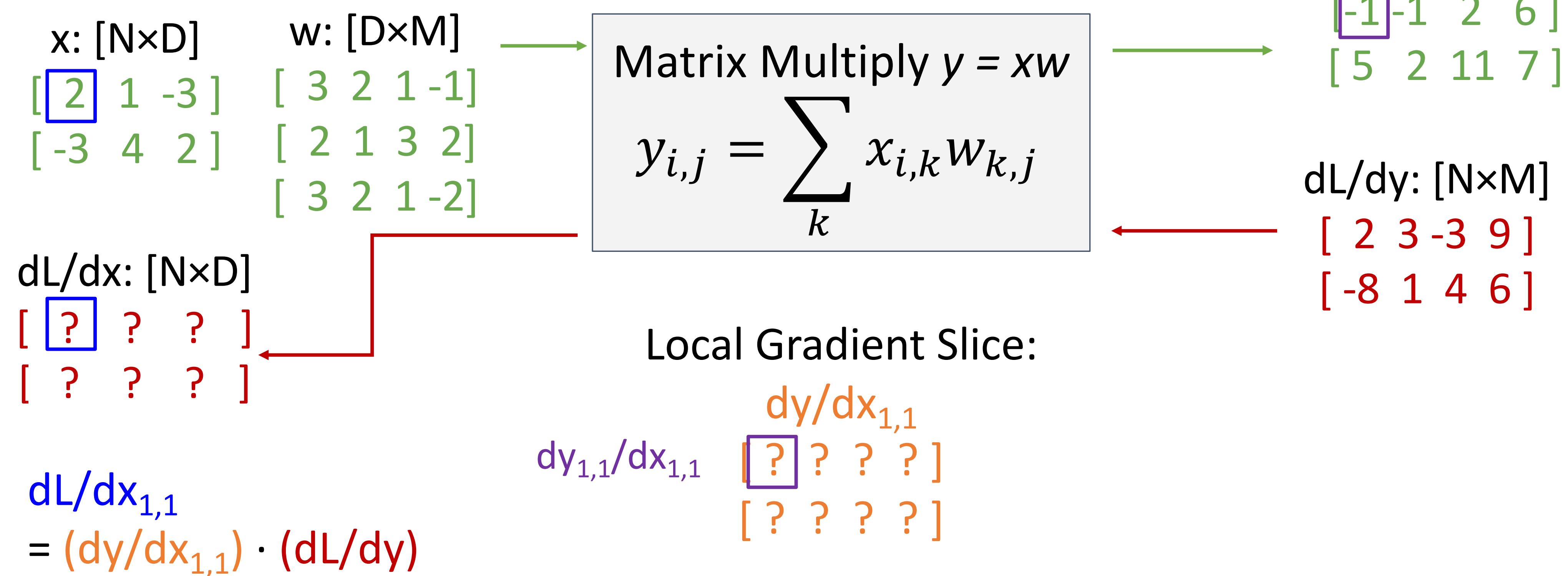
$$N=64, D=M=4096$$

Each Jacobian takes 256 GB of memory! Must work with them implicitly!

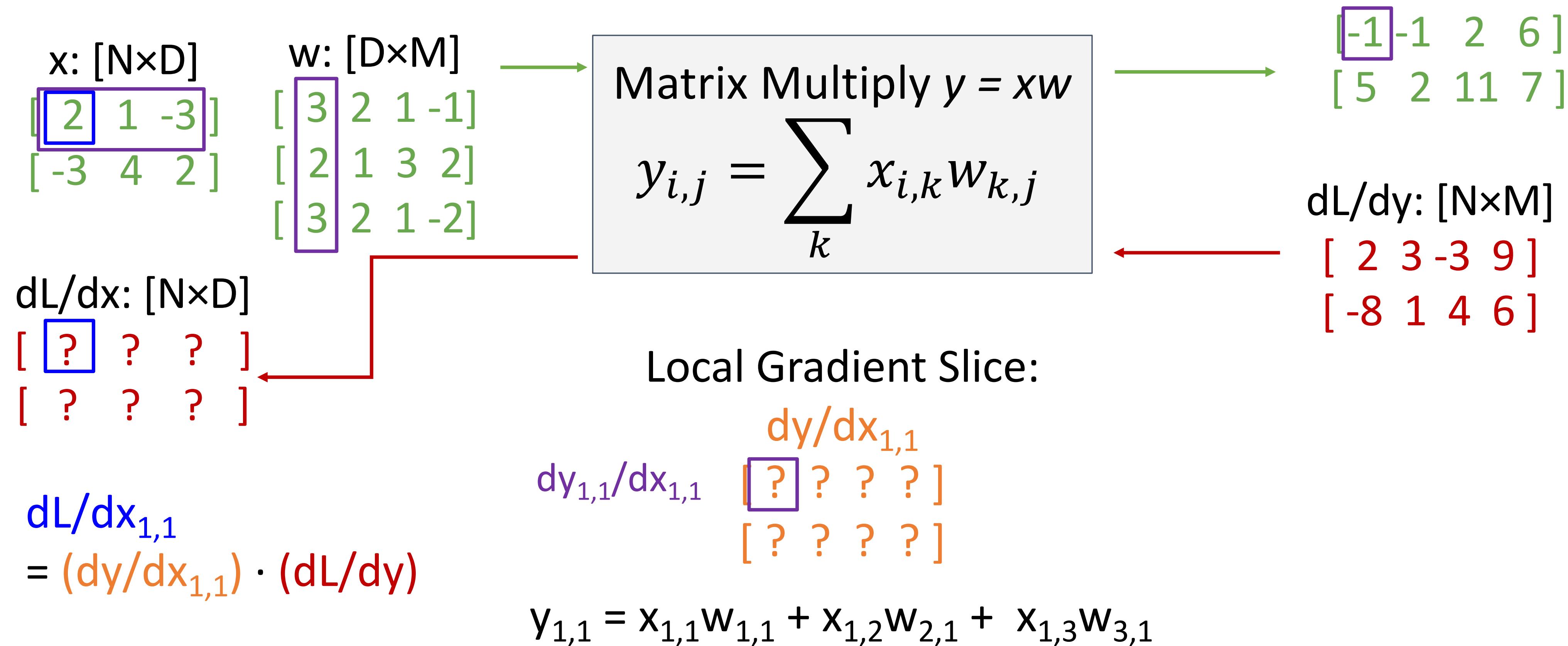
Example: Matrix Multiplication



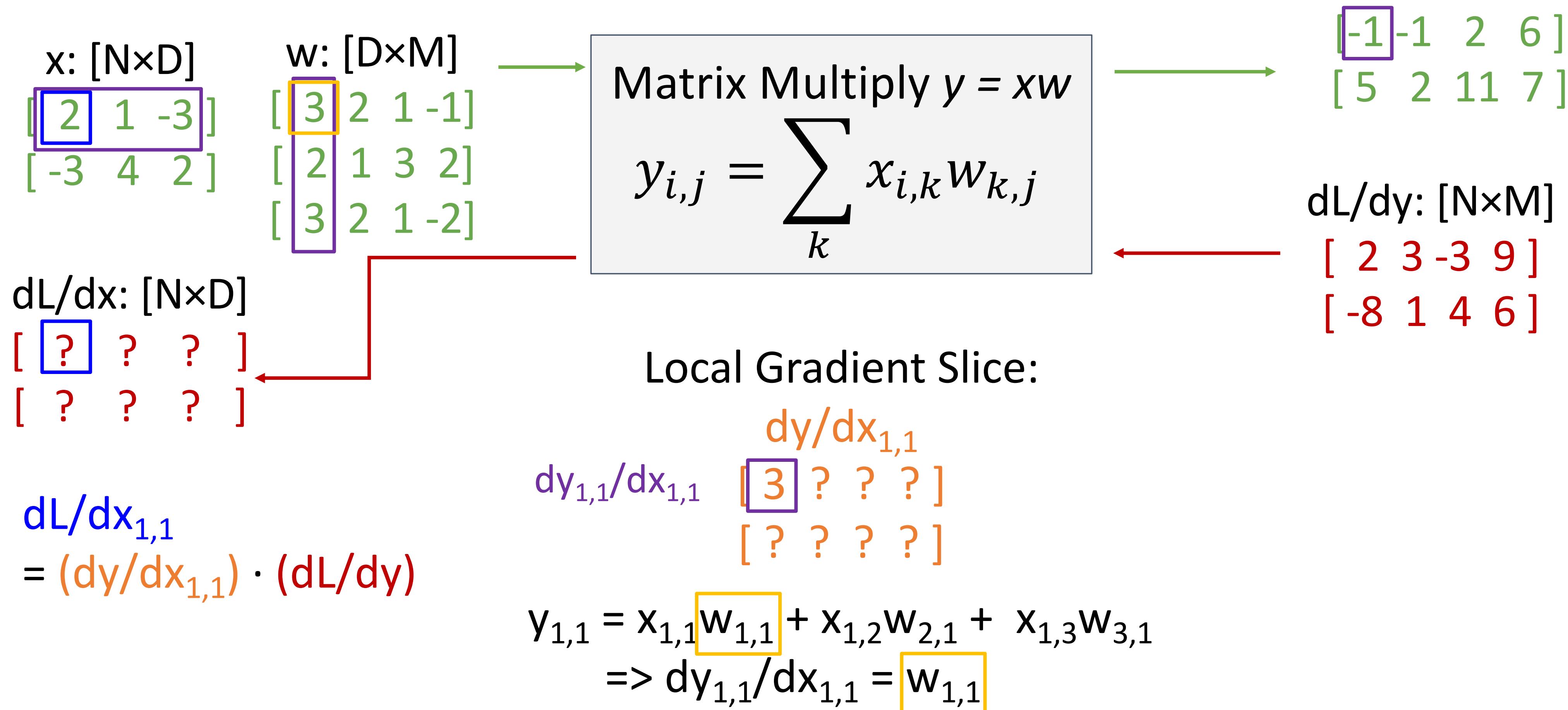
Example: Matrix Multiplication



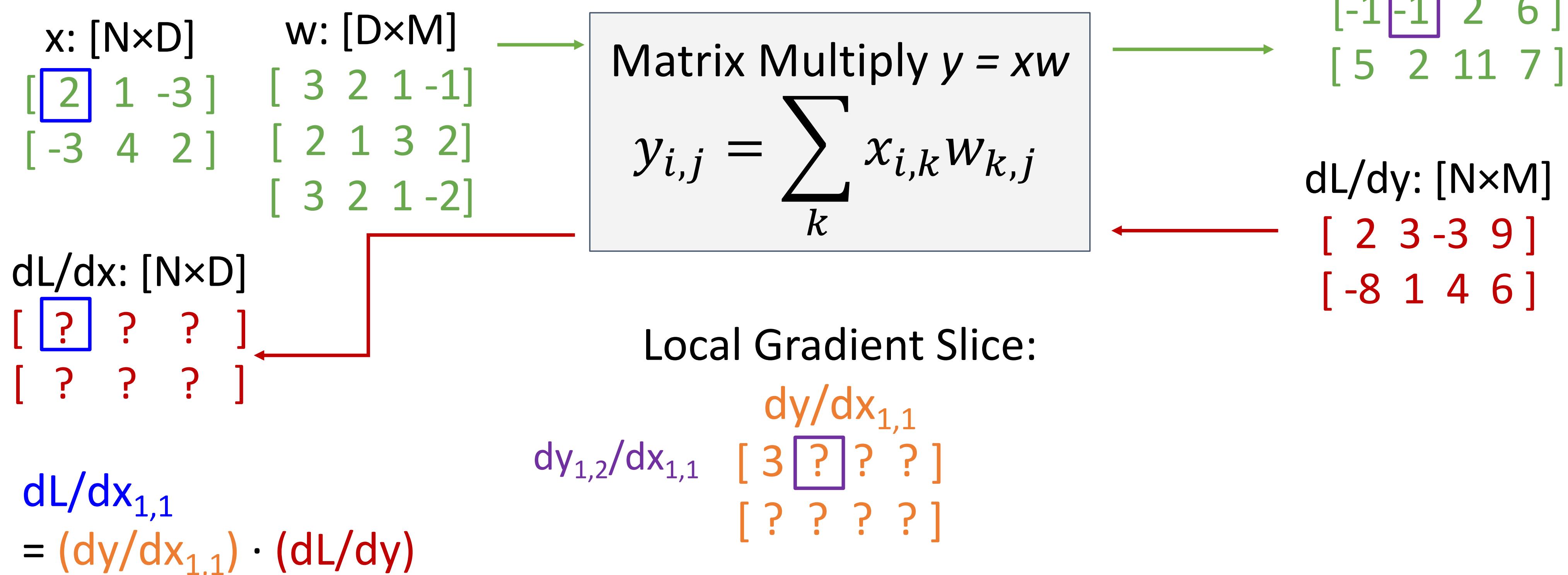
Example: Matrix Multiplication



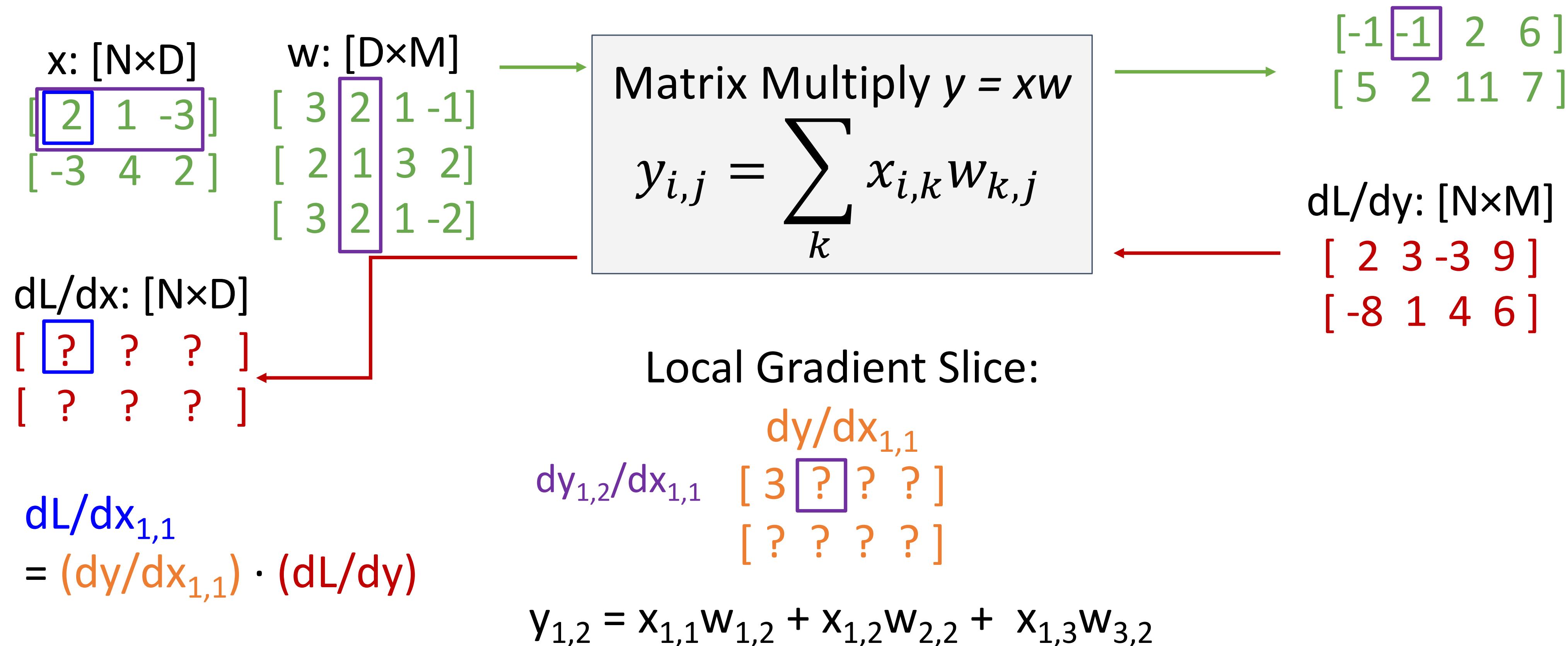
Example: Matrix Multiplication



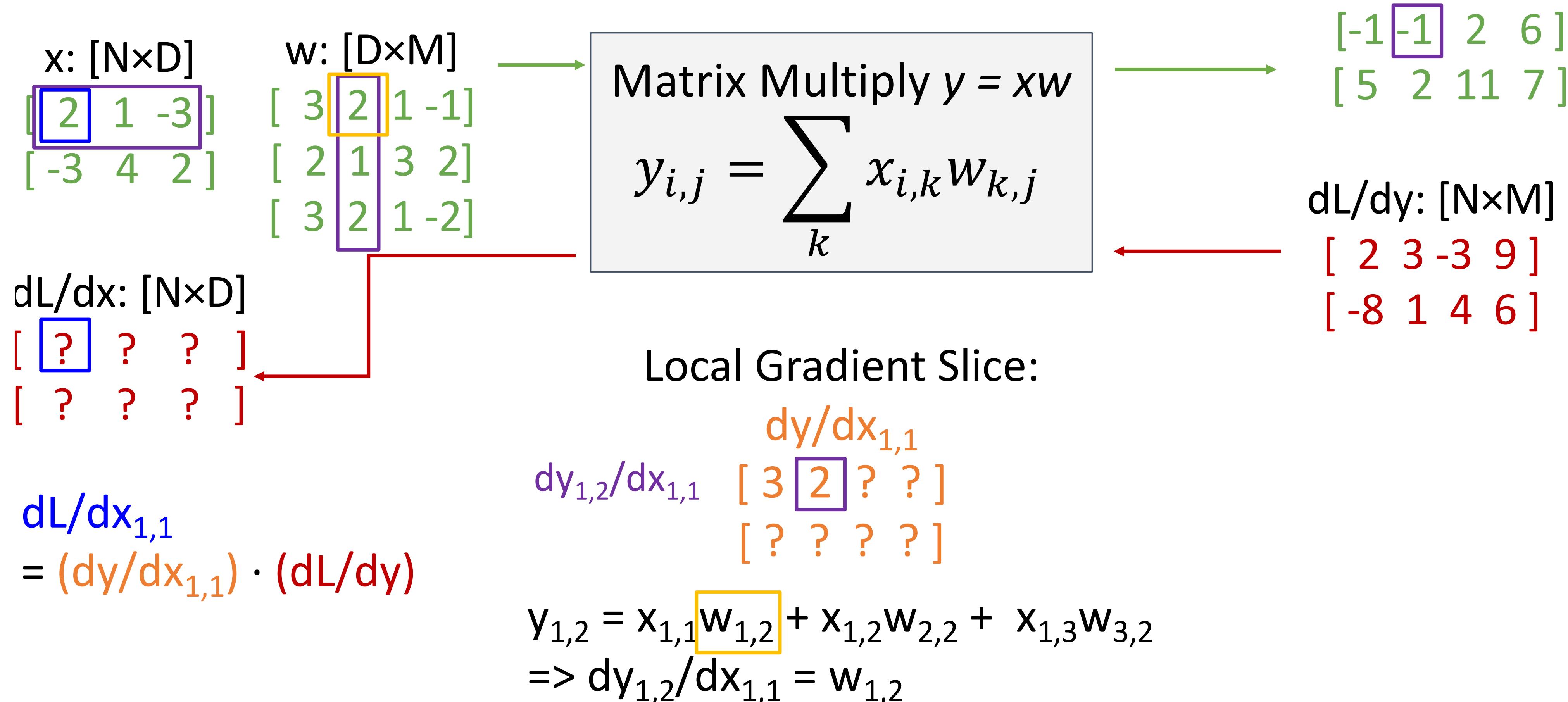
Example: Matrix Multiplication



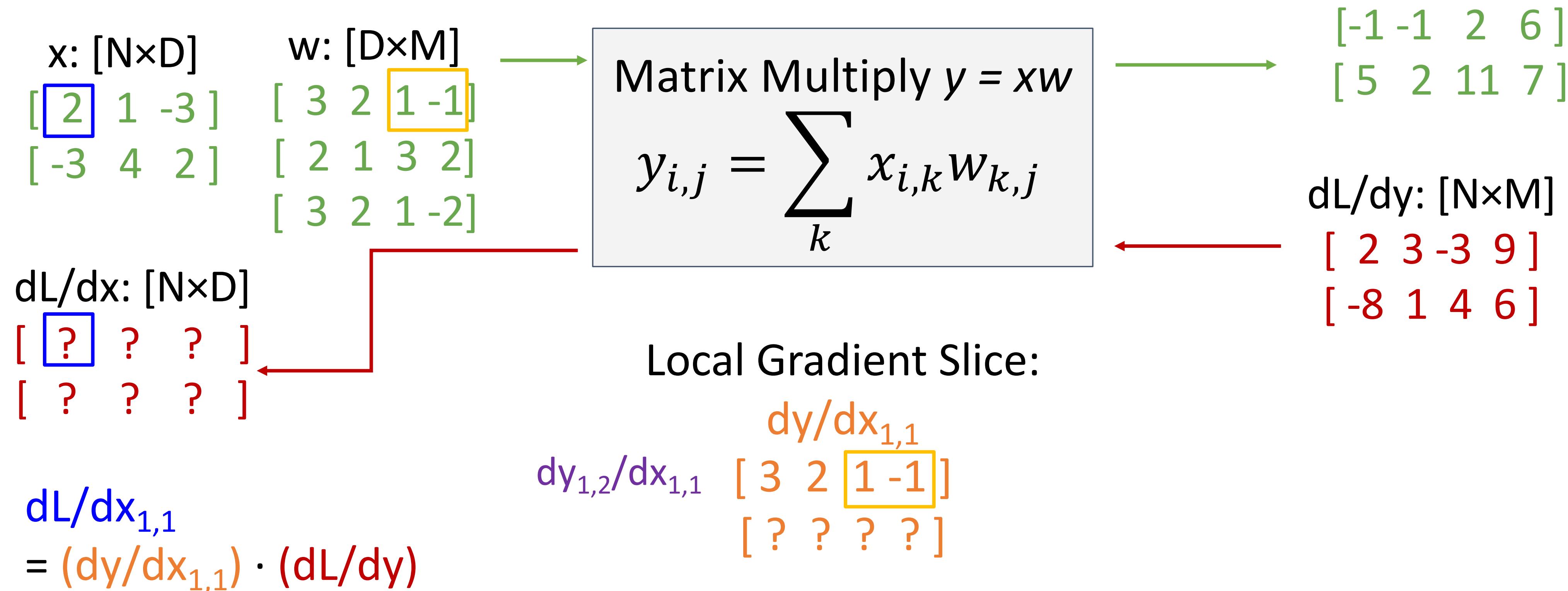
Example: Matrix Multiplication



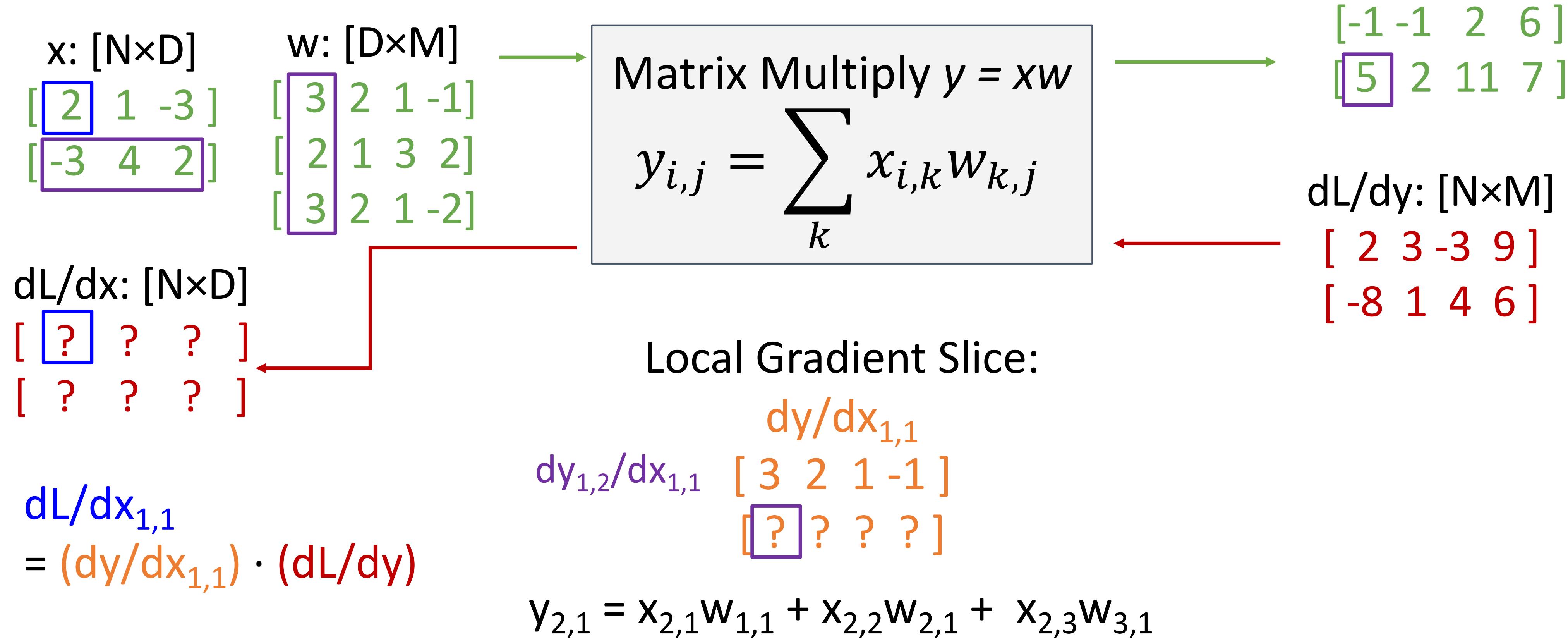
Example: Matrix Multiplication



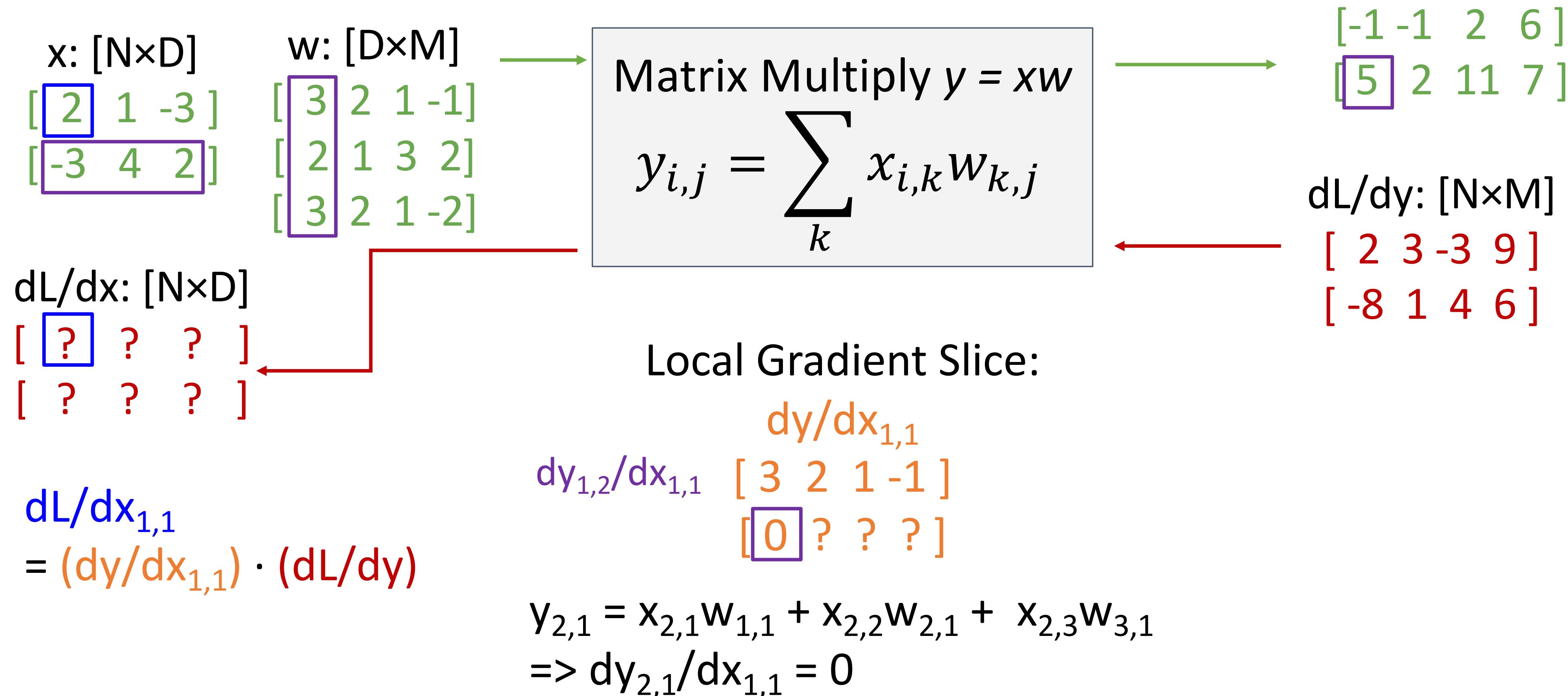
Example: Matrix Multiplication



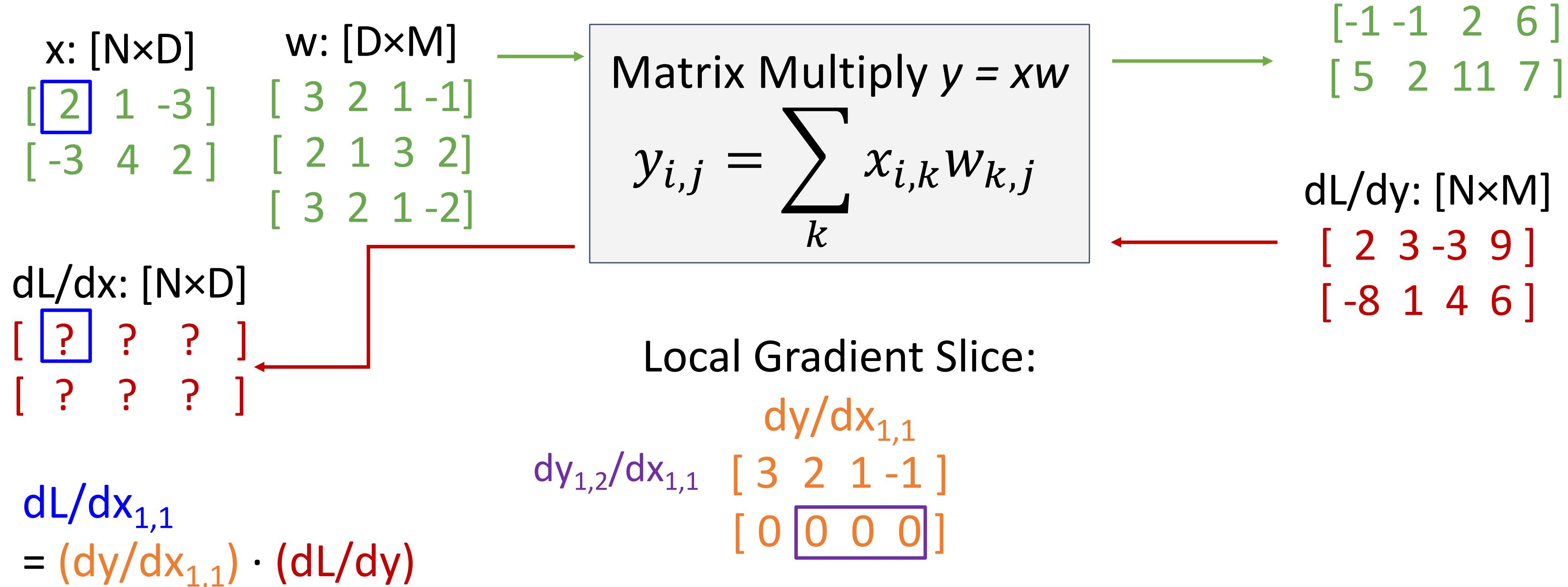
Example: Matrix Multiplication



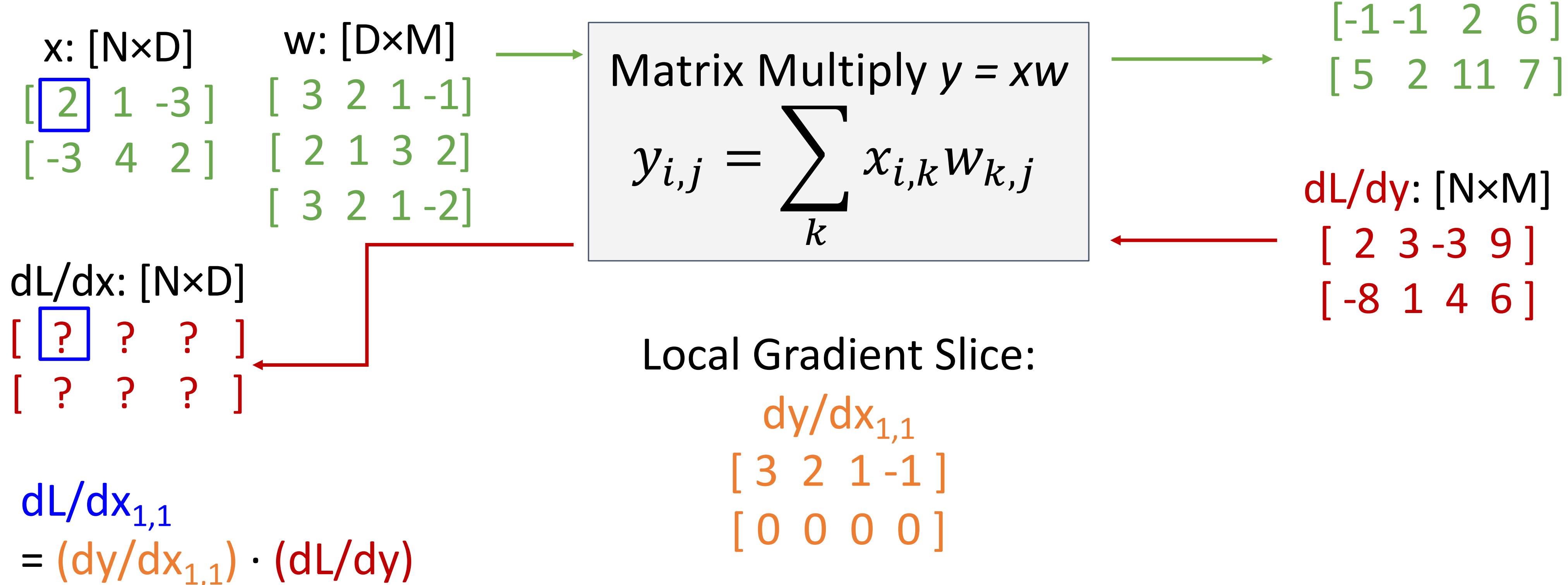
Example: Matrix Multiplication



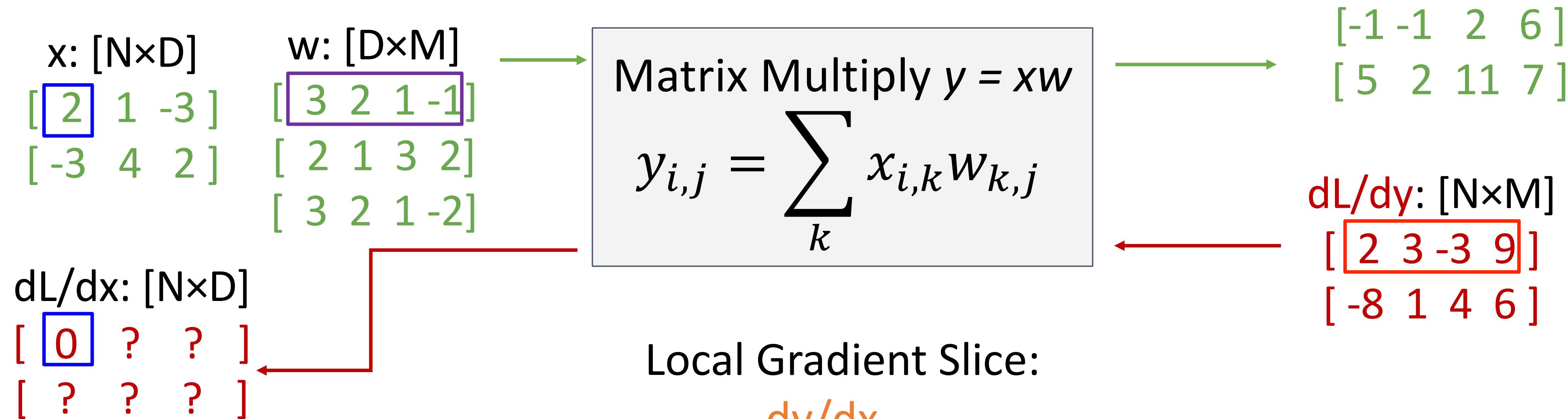
Example: Matrix Multiplication



Example: Matrix Multiplication

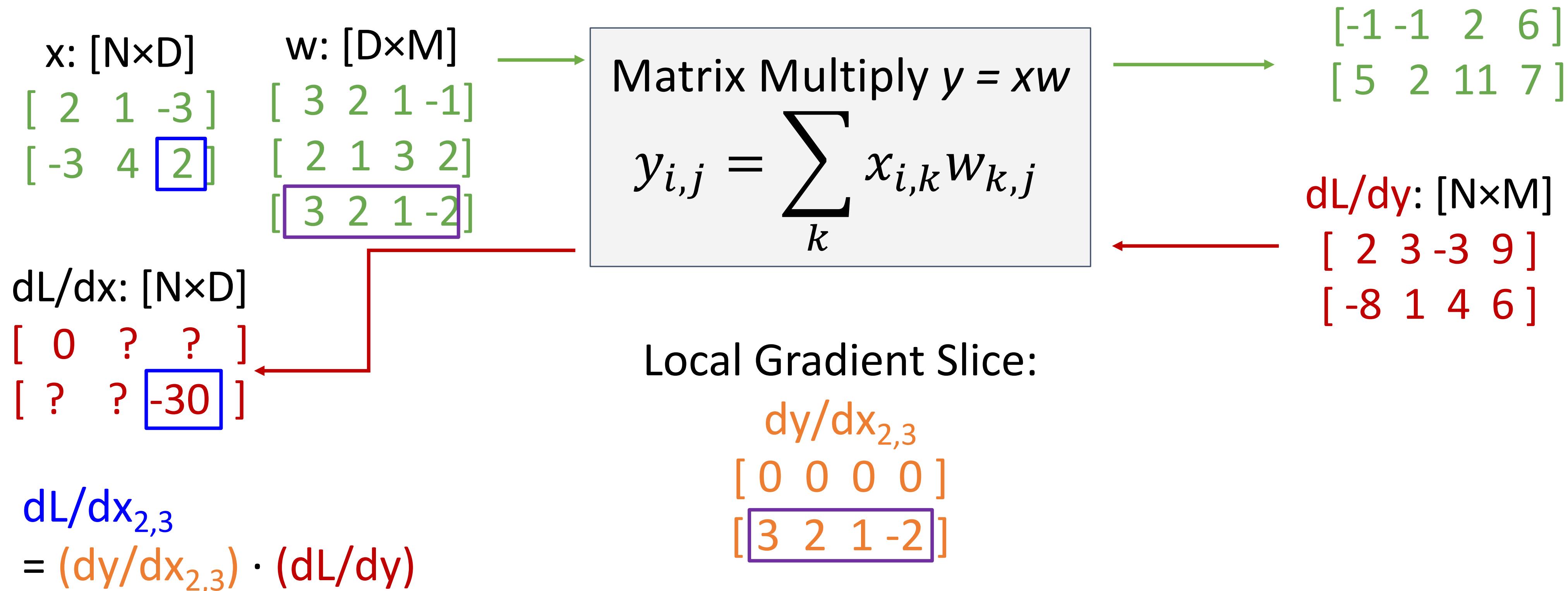


Example: Matrix Multiplication

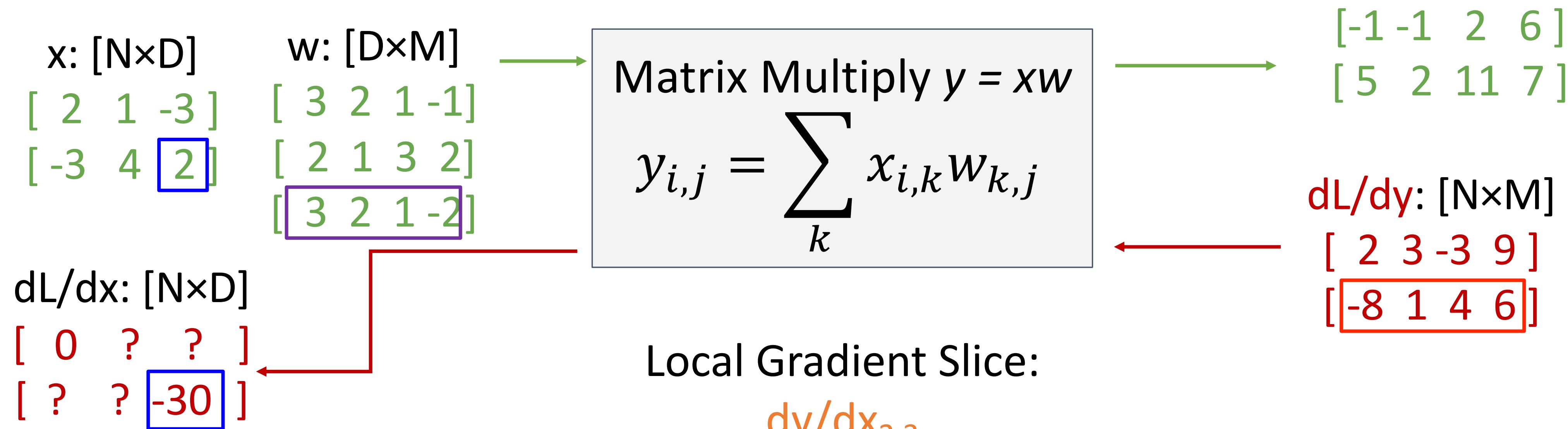


$$\begin{aligned}
 dL/dx_{1,1} &= (dy/dx_{1,1}) \cdot (dL/dy) \\
 &= (w_{1,:}) \cdot (dL/dy_{1,:}) \\
 &= 3*2 + 2*3 + 1*(-3) + (-1)*9 = 0
 \end{aligned}$$

Example: Matrix Multiplication



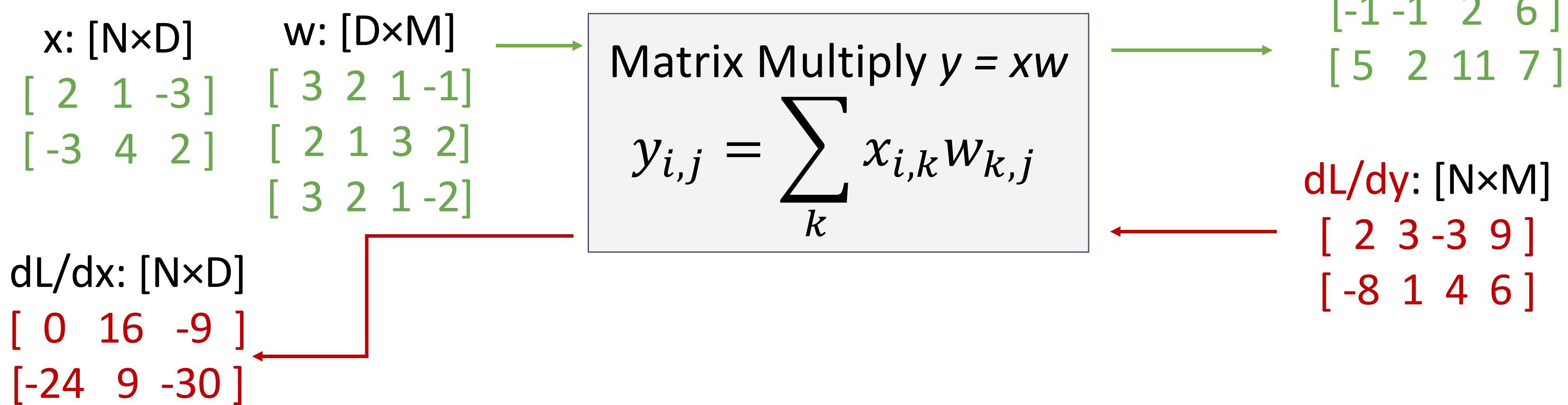
Example: Matrix Multiplication



$$\begin{aligned}
 dL/dx_{2,3} &= (\frac{dy}{dx}_{2,3}) \cdot (dL/dy) \\
 &= (w_{3,:}) \cdot (dL/dy_{2,:}) \\
 &= 3*(-8) + 2*1 + 1*4 + (-2)*6 = -30
 \end{aligned}$$

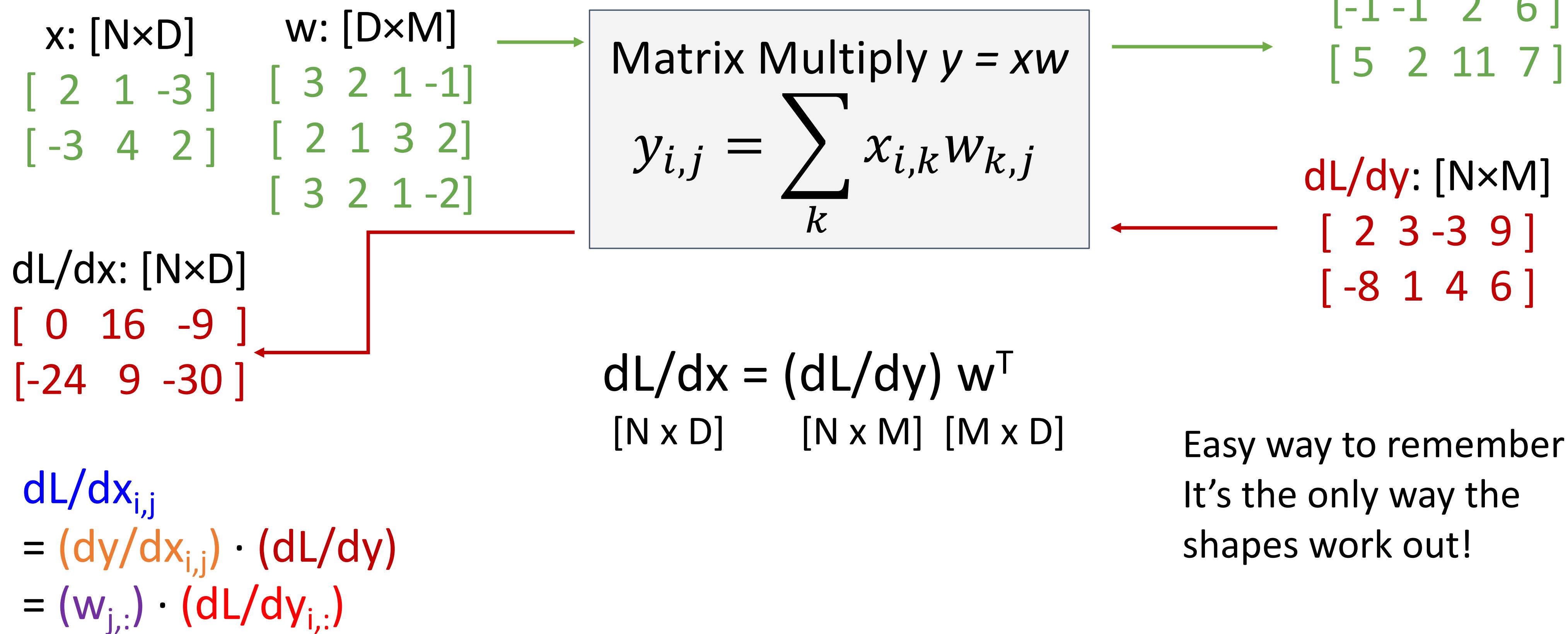


Example: Matrix Multiplication

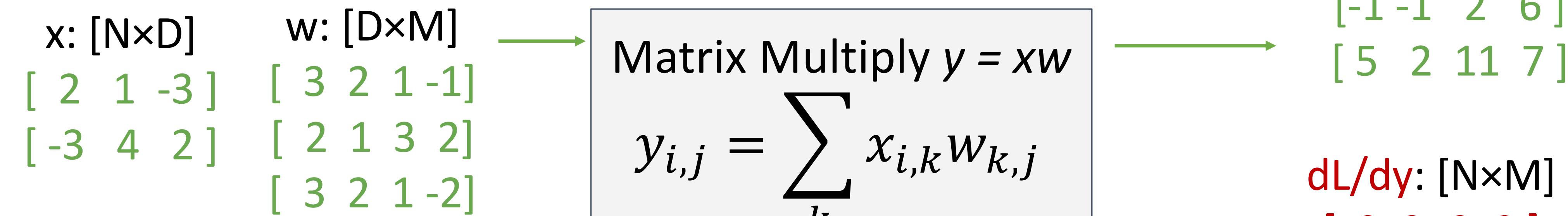


$$\begin{aligned}
 dL/dx_{i,j} &= (dy/dx_{i,j}) \cdot (dL/dy) \\
 &= (w_{j,:}) \cdot (dL/dy_{i,:})
 \end{aligned}$$

Example: Matrix Multiplication



Example: Matrix Multiplication



$$dL/dx = (dL/dy) w^T$$

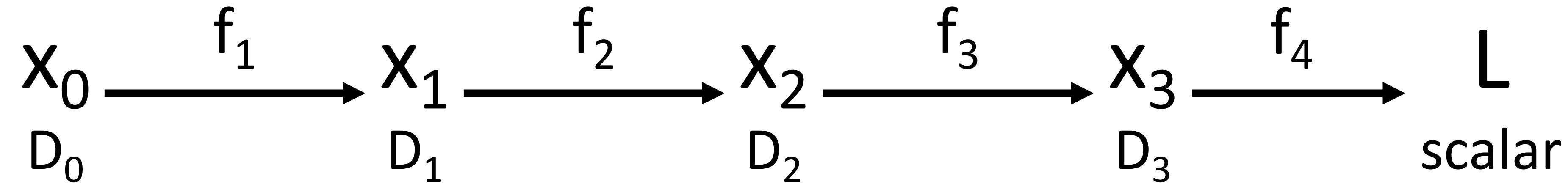
$[N \times D] \quad [N \times M] \quad [M \times D]$

$$dL/dw = x^T (dL/dy)$$

$[D \times M] \quad [D \times N] \quad [N \times M]$

Easy way to remember:
It's the only way the
shapes work out!

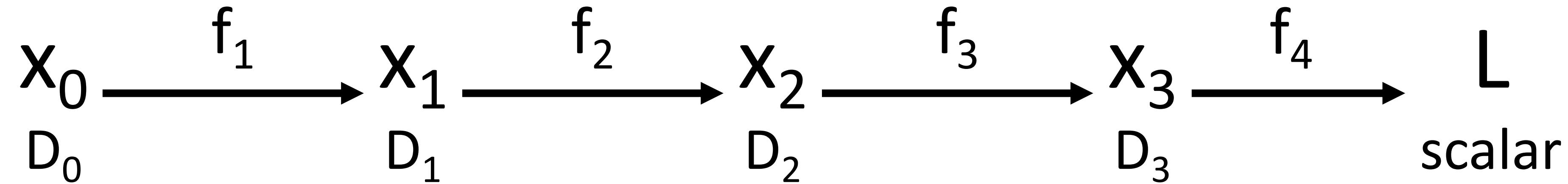
Backpropagation: Another View



Chain rule

$$\frac{\partial L}{\partial x_0} = \left(\frac{\partial x_1}{\partial x_0} \right) \left(\frac{\partial x_2}{\partial x_1} \right) \left(\frac{\partial x_3}{\partial x_2} \right) \left(\frac{\partial L}{\partial x_3} \right)$$

Backpropagation: Another View



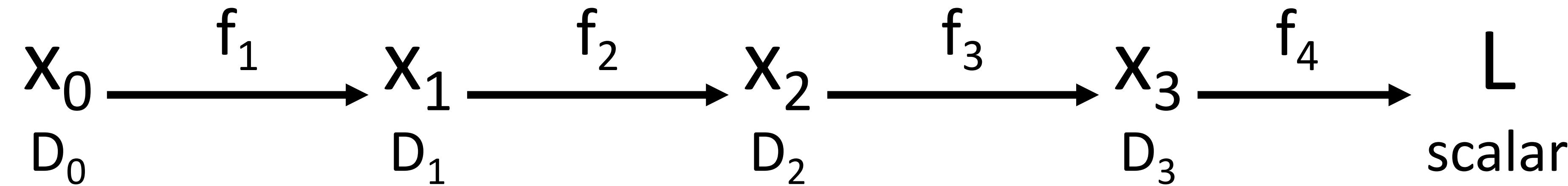
Matrix multiplication is **associative**: we can compute products in any order

Chain rule

$$\frac{\partial L}{\partial x_0} = \left(\frac{\partial x_1}{\partial x_0} \right) \left(\frac{\partial x_2}{\partial x_1} \right) \left(\frac{\partial x_3}{\partial x_2} \right) \left(\frac{\partial L}{\partial x_3} \right)$$

$[D_0 \times D_1] [D_1 \times D_2] [D_2 \times D_3] [D_3]$

Reverse-Mode Automatic Differentiation



Matrix multiplication is **associative**: we can compute products in any order

Computing products right-to-left avoids matrix-matrix products; only needs matrix-vector

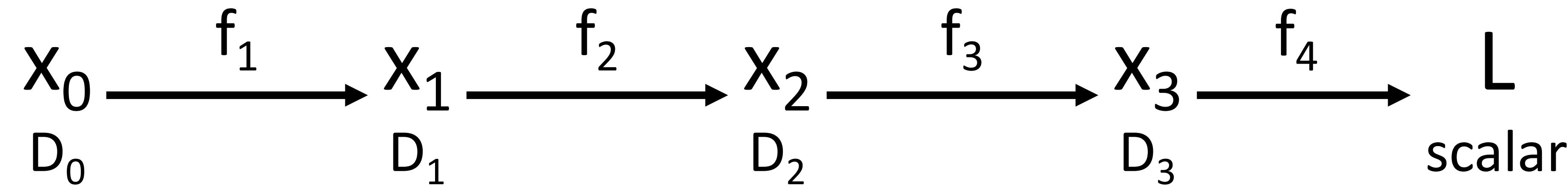
←

Chain rule

$$\frac{\partial L}{\partial x_0} = \left(\frac{\partial x_1}{\partial x_0} \right) \left(\frac{\partial x_2}{\partial x_1} \right) \left(\frac{\partial x_3}{\partial x_2} \right) \left(\frac{\partial L}{\partial x_3} \right)$$

$$[D_0 \times D_1] \ [D_1 \times D_2] \ [D_2 \times D_3] \ [D_3]$$

Reverse-Mode Automatic Differentiation



Matrix multiplication is **associative**: we can compute products in any order

Computing products right-to-left avoids matrix-matrix products; only needs matrix-vector

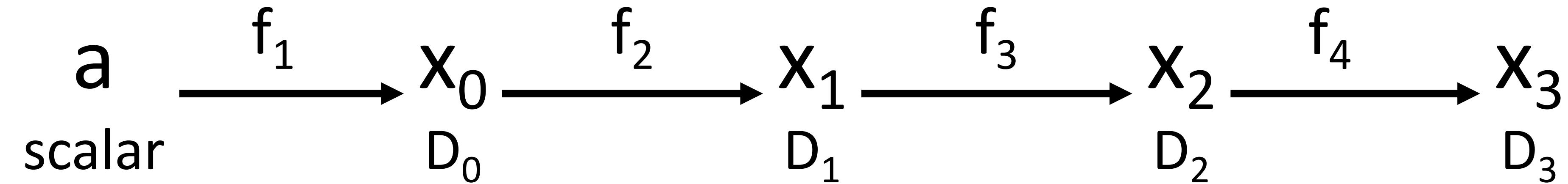
$$\text{Chain rule} \quad \frac{\partial L}{\partial x_0} = \left(\frac{\partial x_1}{\partial x_0} \right) \left(\frac{\partial x_2}{\partial x_1} \right) \left(\frac{\partial x_3}{\partial x_2} \right) \left(\frac{\partial L}{\partial x_3} \right)$$

Compute grad of scalar output
w/respect to all vector inputs

$[D_0 \times D_1] [D_1 \times D_2] [D_2 \times D_3] [D_3]$

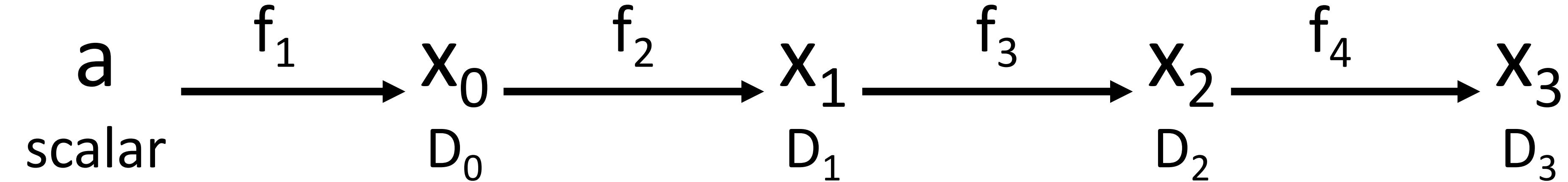
What if we want
grads of scalar
input w/respect
to vector
outputs?

Forward-Mode Automatic Differentiation



Chain rule
$$\frac{\partial x_3}{\partial a} = \left(\frac{\partial x_0}{\partial a} \right) \left(\frac{\partial x_1}{\partial x_0} \right) \left(\frac{\partial x_2}{\partial x_1} \right) \left(\frac{\partial x_3}{\partial x_2} \right)$$
$$[D_0] \quad [D_0 \times D_1] \quad [D_1 \times D_2] \quad [D_2 \times D_3]$$

Forward-Mode Automatic Differentiation



Computing products left-to-right avoids matrix-matrix products; only needs matrix-vector

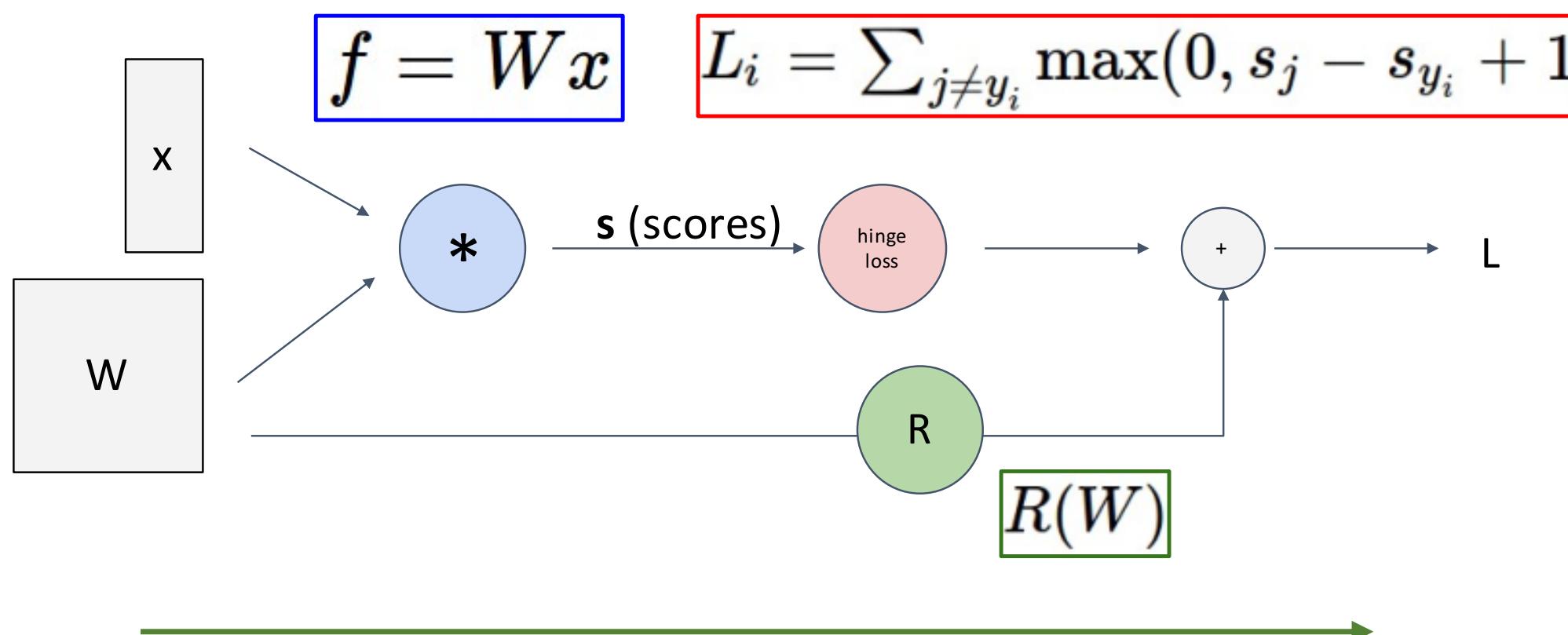
Chain rule

$$\frac{\partial x_3}{\partial a} = \overbrace{\left(\frac{\partial x_0}{\partial a} \right) \left(\frac{\partial x_1}{\partial x_0} \right) \left(\frac{\partial x_2}{\partial x_1} \right) \left(\frac{\partial x_3}{\partial x_2} \right)}^{\longrightarrow}$$

$[D_0] \quad [D_0 \times D_1] \quad [D_1 \times D_2] \quad [D_2 \times D_3]$

Summary

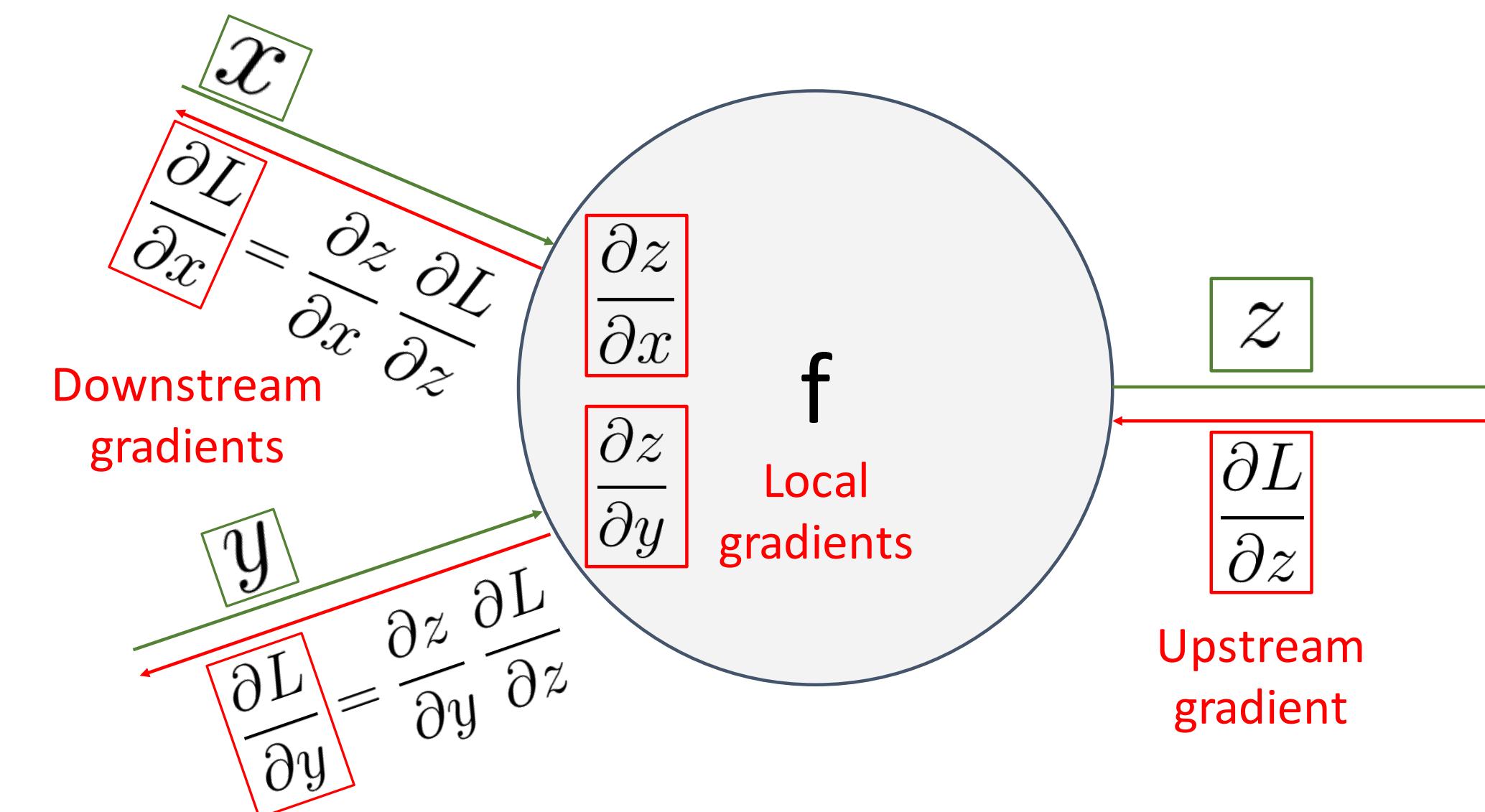
Represent complex expressions as **computational graphs**



Forward pass computes outputs

Backward pass computes gradients

During the backward pass, each node in the graph receives **upstream gradients** and multiplies them by **local gradients** to compute **downstream gradients**



Summary

Backprop can be implemented with “flat” code where the backward pass looks like forward pass reversed (Use this for A2!)

```
def f(w0, x0, w1, x1, w2):
    s0 = w0 * x0
    s1 = w1 * x1
    s2 = s0 + s1
    s3 = s2 + w2
    L = sigmoid(s3)

    grad_L = 1.0
    grad_s3 = grad_L * (1 - L) * L
    grad_w2 = grad_s3
    grad_s2 = grad_s3
    grad_s0 = grad_s2
    grad_s1 = grad_s2
    grad_w1 = grad_s1 * x1
    grad_x1 = grad_s1 * w1
    grad_w0 = grad_s0 * x0
    grad_x0 = grad_s0 * w0
```

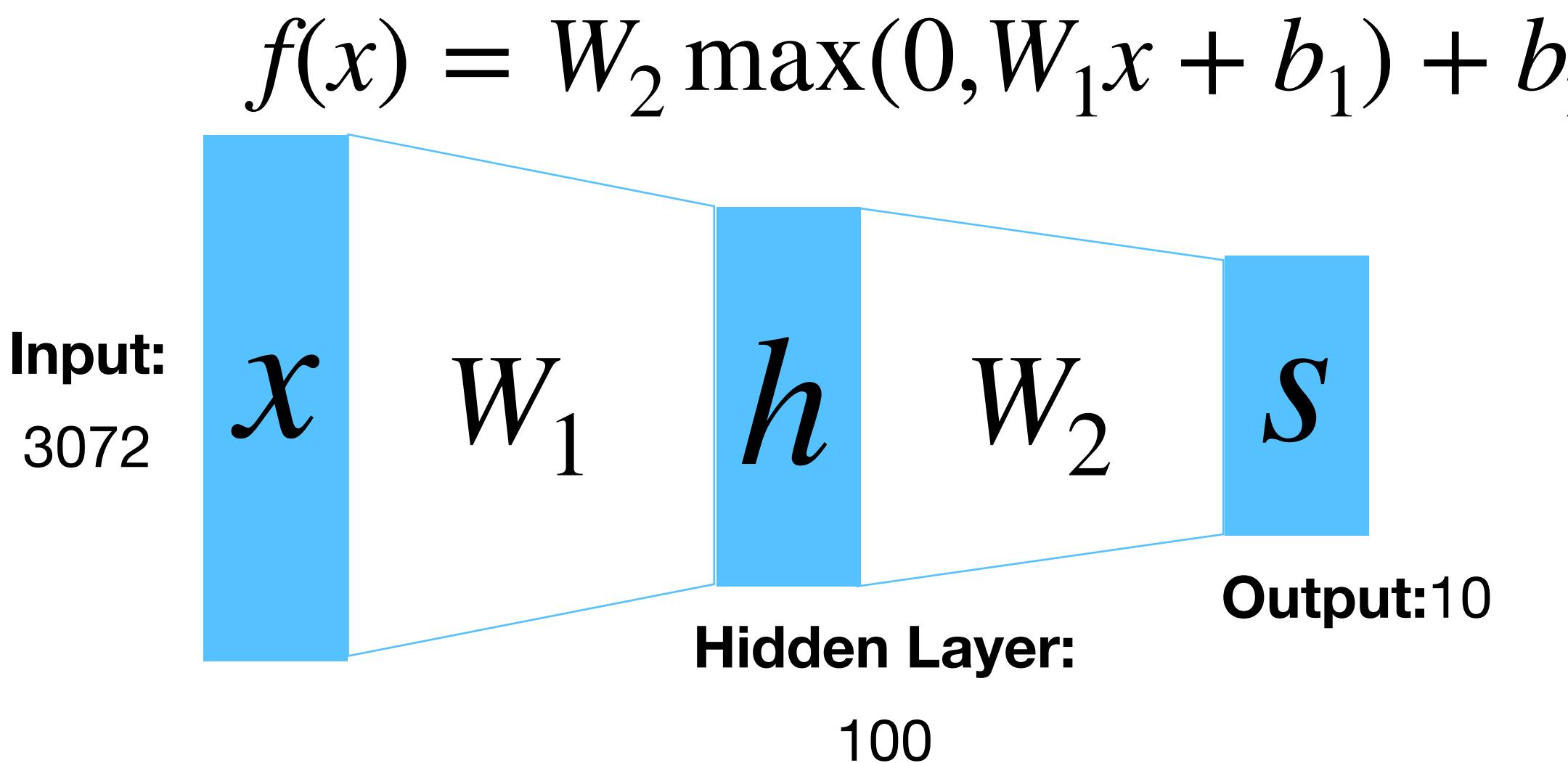
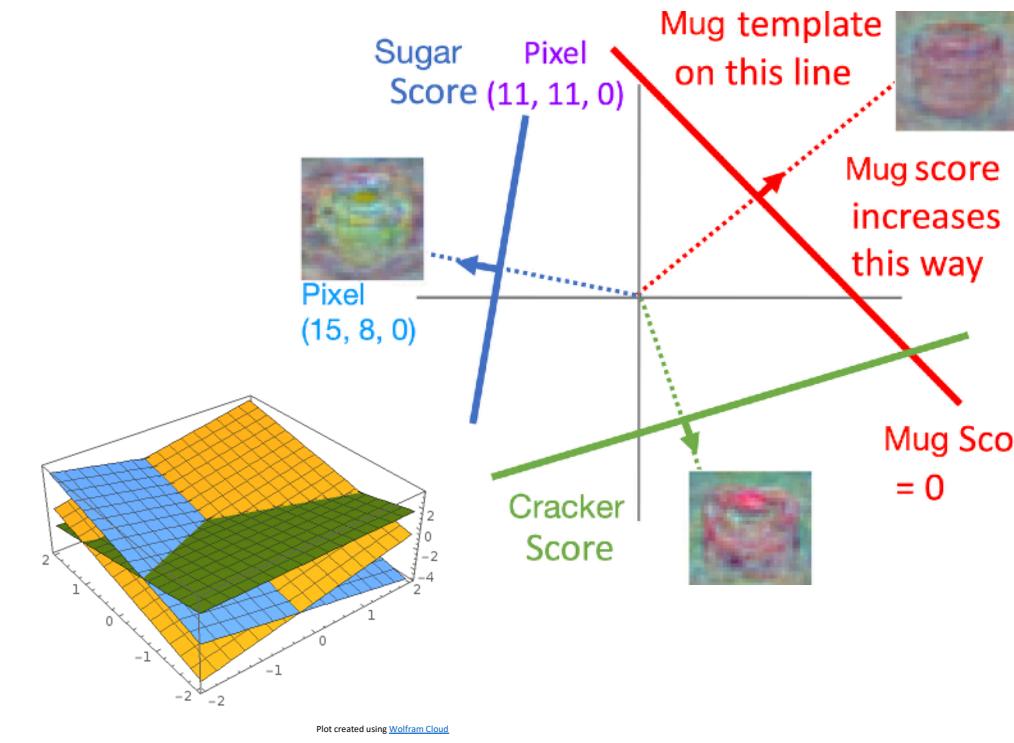
Backprop can be implemented with a modular API, as a set of paired forward/backward functions

```
class Multiply(torch.autograd.Function):
    @staticmethod
    def forward(ctx, x, y):
        ctx.save_for_backward(x, y)
        z = x * y
        return z

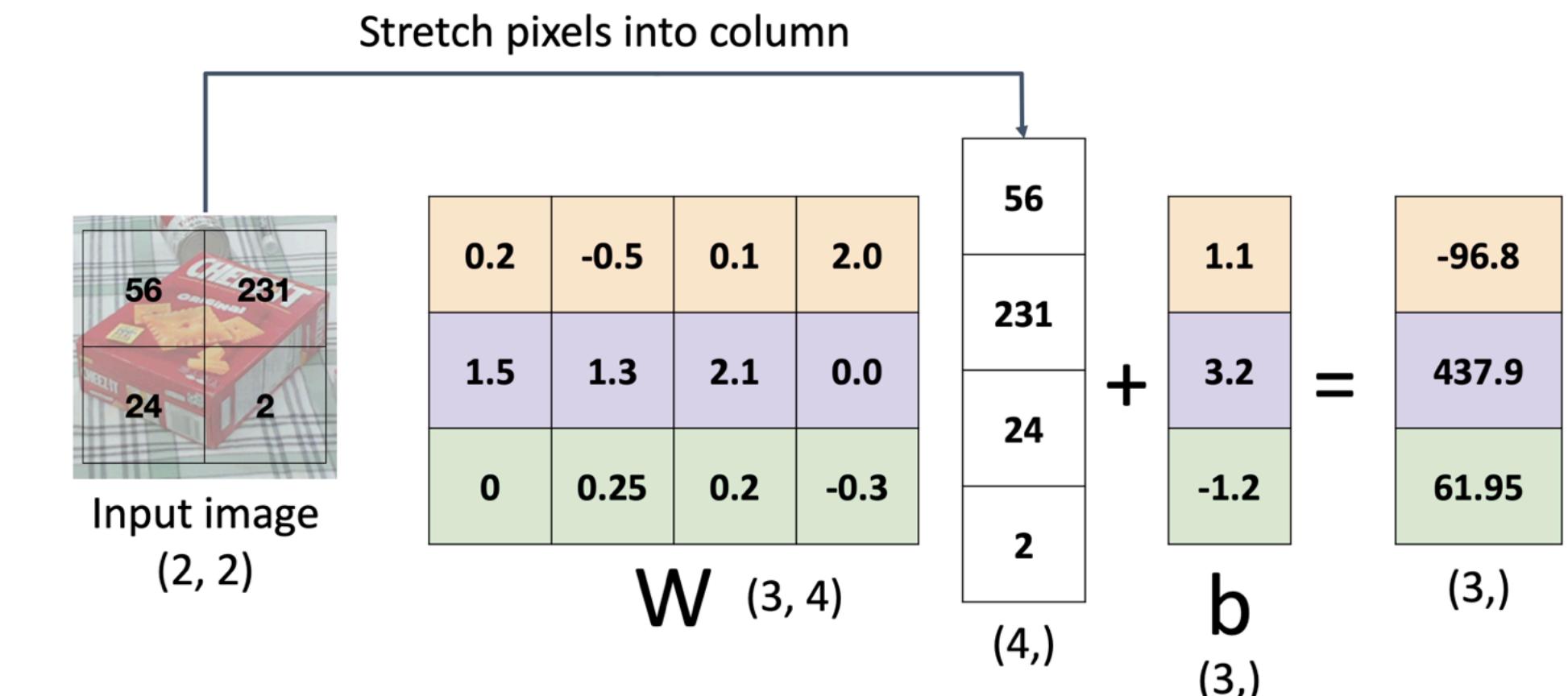
    @staticmethod
    def backward(ctx, grad_z):
        x, y = ctx.saved_tensors
        grad_x = y * grad_z    # dz/dx * dL/dz
        grad_y = x * grad_z    # dz/dy * dL/dz
        return grad_x, grad_y
```



Summary



Problem: So far our classifiers don't respect the spatial structure of images!



Next time: Convolutional Neural Networks



DR

DeepRob

Lecture 6
Backpropagation
University of Michigan and University of Minnesota

$$\frac{\partial L}{\partial W_{\ell_1}}$$

$$\frac{\partial L}{\partial W_{\ell_2}}$$

$$\frac{\partial L}{\partial W_{\ell_3}}$$

$$\frac{\partial L}{\partial W_{\ell_4}}$$

$$\frac{\partial L}{\partial W_{\ell_5}}$$

$$\frac{\partial L}{\partial \text{Out}}$$

